(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号
特開2001－43176
（Ｐ２００１－４３１７６Ａ）
(43)公開日　平成13年２月16日(2001.2.16)

(51)Int.Cl.[7]	識別記号	ＦＩ		ﾃｰﾏｺｰﾄﾞ（参考）
G06F 13/00	620	G06F 13/00	620	

審査請求　未請求　請求項の数10　ＯＬ　（全38頁）

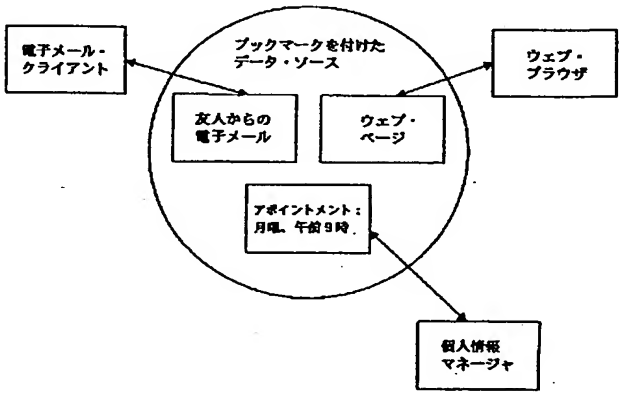
(21)出願番号　特願2000－176806（Ｐ2000－176806）
(22)出願日　平成12年６月13日(2000.6.13)
(31)優先権主張番号　０９／３３３４２１
(32)優先日　平成11年６月15日(1999.6.15)
(33)優先権主張国　米国（ＵＳ）

(71)出願人　591064003
サン・マイクロシステムズ・インコーポレーテッド
SUN MICROSYSTEMS, INCORPORATED
アメリカ合衆国　94303　カリフォルニア州・パロ　アルト・サン　アントニオ　ロード・901
901 SAN ANTONIO ROAD PALO ALTO, CA 94303, U.S.A.
(74)代理人　100064621
弁理士　山川　政樹

最終頁に続く

(54)【発明の名称】小フットプリント・デバイス・アプリケーション用の高性能ブックマーク

(57)【要約】
【課題】　高性能携帯電話、携帯情報端末などの小フットプリント・デバイスのユーザが、様々なタイプのオブジェクトおよび／またはデータ・ソースを参照するブックマークを作成し、格納することができるシステムを提供する。
【解決手段】　各ブックマークは、ＵＲＬを使用してオブジェクト／データ・ソースを参照する。このブックマーク・システムはオープン・エンドにすることができ、実質的にいかなるタイプのオブジェクトまたはデータ・ソースにもブックマークを付けることができる。後にユーザまたはアプリケーションがブックマークを使用してそれぞれのデータ・ソースを参照し、表示や編集など、あるタイプのアクションをそのデータ・ソースに対して実行することができる。



1

【特許請求の範囲】

【請求項１】　データ・ソースへの持続的参照を作成するシステムであって、

処理ユニットおよびシステム・メモリを含む小フットプリント・デバイスと、

前記システム・メモリに格納され、コンピューティング・サービスを実装するプログラム・モジュールをサポートするソフトウェア・フレームワークと、

１つまたは複数の前記プログラム・モジュールに実装され、ユーザが前記データ・ソースを選択することに応答してデータ・ソースへの持続的参照を作成するように動作可能である第１のコンピューティング・サービスと、

１つまたは複数の前記プログラム・モジュールに実装され、前記持続的参照を使用して前記データ・ソースにアクセスするように動作可能である第２のコンピューティング・サービスとを含むシステム。

【請求項２】　前記データ・ソースをカプセル化するエンティティを作成するように動作可能な起動フレームワークをさらに含み、第１のコンピューティング・サービスによって作成された前記持続的参照は、前記データ・ソースをカプセル化する前記エンティティを参照する請求項１に記載のシステム。

【請求項３】　前記起動フレームワークがさらに、前記第２のコンピューティング・サービスを呼び出して、前記データ・ソースに対して操作を実行するように動作可能である請求項２に記載のシステム。

【請求項４】　前記起動フレームワークがＪａｖａＢｅａｎｓ起動フレームワークである請求項３に記載のシステム。

【請求項５】　前記ソフトウェア・フレームワークが、３００キロバイト未満のメモリを必要とする請求項１に記載のシステム。

【請求項６】　前記小フットプリント・デバイスが、携帯情報端末（ＰＤＡ）、携帯電話、および全世界測位システム（ＧＰＳ）受信機からなるグループからの小フットプリント・デバイスである請求項１に記載のシステム。

【請求項７】　前記小フットプリント・デバイスが、ゲーム・コンソール、ウェアラブル・コンピューティング・デバイス、セット・トップ・ボックス、および電子ブック・デバイスからなるグループからの小フットプリント・デバイスである請求項１に記載のシステム。

【請求項８】　前記小フットプリント・デバイスが、２メガバイト未満のメモリを含む請求項１に記載のシステム。

【請求項９】　前記小フットプリント・デバイスが、２０平方インチより小さいディスプレイ画面を含む請求項１に記載のシステム。

【請求項１０】　前記小フットプリント・デバイスが、専らバッテリ駆動型である請求項１に記載のシステム。

2

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、一般にコンピュータ・アプリケーション・プログラムおよび小さなフットプリントのデバイスに関する。より詳細には、本発明は、その小フットプリント・デバイス上で動作するアプリケーションおよびサービスによって使用するための様々なタイプのデータ・ソース・オブジェクトにブックマークを付けるシステムおよび方法を含む。

【０００２】

【従来の技術】「高性能」の小フットプリント・デバイスの分野は、急速に成長し、変化している。小フットプリント・デバイスとしては、ハンドヘルド・コンピュータ、携帯情報端末（ＰＤＡ）、携帯電話、全世界測位システム（ＧＰＳ）受信機、ゲーム・コンソール、およびさらに多数のこのようなデバイスなどがある。これらのデバイスは、よりインテリジェントにかつ相互接続されるようになりつつある。Ｓｕｎ Ｍｉｃｒｏｓｙｓｔｅｍｓ社のＪｉｎｉ™などの技術およびＯｐｅｎＳｅｒｖｉｃｅ Ｇａｔｅｗａｙ Ｉｎｉｔｉａｔｉｖｅ（ＯＳＧＩ）などのイニシアチブが、従来のコンピュータ・ネットワークの概念を、小フットプリント・デバイスを含めるように拡張している。

【０００３】このデバイス相互接続の増加によって、新しいタイプのコンピューティング・サービスと、デバイス間ベースおよびデバイス内ベースのサービスのコンピューティング・サービスを統合する新しい方法に対する必要性が出てきた。「サービス」とは、デバイス内あるいはデバイスからアクセス可能に実装されたエンティティであり、人、アプリケーションまたは別のサービスによって使用することができるものである。サービスの概念は幅広く、多数の異なる尺度で考慮することができる。たとえば、サービスとしては、共有印刷、電子メール、テレフォニーなど、周知のネットワーク・ベースのサービスがある。サービスには、ローカル・ネットワーク内のデバイスの電力消費量を制御することができるエネルギー管理サービス、エラーが生じたときにデバイスが情報をサービス技術者に送信することができる診断サービス、保健専門家に急患を即座に通知する健康監視サービスなど、それほど周知でない例もある。

【０００４】サービスには、ローカルのマシンまたはデバイス内に位置する、実行可能なモジュールまたはアプリケーションなどもある。たとえば、ローカル・アプリケーション・プログラムは、サービスを利用して、ＨＴＴＰサーバ、ＨＴＭＬレンダ・エンジン・サービス、ブックマーク・サービス、ユーザ・インタフェース・サービスなどと通信することができる。この例では、アプリケーション・プログラムがこれらのサービスを共に使用して、ウェブ・ブラウザ・プログラムを実装することができる。

【０００５】今日、多数のサービスおよびアプリケーションを共に単一の小フットプリント・デバイスで実行することが、より一般的になりつつある。しかし、メモリ、処理能力および他のリソースが、一般に小フットプリント・デバイスでは非常に制限されるため、専用の軽量サービス／アプリケーション・コンテインメント・フレームワークが、所望のサービスおよびアプリケーションの統合を達成するために必要である。コンテインメント・フレームワークは、いかなる種類の小フットプリント・デバイス用のいかなるタイプのサービスおよびアプリケーション用のフレームワークをも提供するために、十分柔軟性があり、拡張可能であることも望ましい。さらなる目標は、コンテインメント・フレームワークが、Ｊｉｎｉ™ネットワークにおけるデバイスに使用可能なサービスなど、オフ・デバイス・サービスと互換性を持ち、統合されるようにすることであることがある。本明細書に記載するコンテインメント・フレームワークは、上述の目標を達成する。

【０００６】軽量のコンテインメント・フレームワークは、携帯情報端末、高性能携帯電話などの小フットプリント・デバイスで、従来はデスクトップ・コンピューティング環境に関連したタイプの多目的アプリケーション・プログラムを実行することができる。たとえば、Ｓｕｎ Ｍｉｃｒｏｓｙｓｔｅｍｓ社から入手可能なＰｅｒｓｏｎａｌ Ａｐｐｌｉｃａｔｉｏｎｓスーツは、コンテインメント・フレームワークの一実施形態を中心に構築されている。Ｐｅｒｓｏｎａｌ Ａｐｐｌｉｃａｔｉｏｎｓスーツは、Ｐｅｒｓｏｎａｌ Ａｐｐｌｉｃａｔｉｏｎｓ Ｂｒｏｗｓｅｒ、Ｐｅｒｓｏｎａｌ Ａｐｐｌｉｃａｔｉｏｎｓ Ｅｍａｉｌ Ｃｌｉｅｎｔおよび Ｐｅｒｓｏｎａｌ Ｏｒｇａｎｉｚｅｒなど、コンパクトなメモリ効率のよいアプリケーションの統合セットを含んでいる。

【０００７】これらのタイプのアプリケーションが小フットプリント・デバイスに使用可能になるので、様々なアプリケーションまたはサービスによって使用あるいは実装されるエンティティ、オブジェクトおよびデータ・ソースを統合および抽象化する汎用機構を提供することが、ますます望ましくなる。このタイプの抽象化および統合を可能にするための一手法は、ブックマーク・サービスを使用可能にして、アプリケーション・プログラムまたはサービスで作成あるいは使用することができる、様々なタイプのエンティティ、オブジェクトおよびデータ・ソースのいずれにも対する持続的参照を提供することである。たとえば、ユーザは、特定の電子メール、またはコンタクト・リストにある特定のコンタクト・エントリにブックマークを付けて、後に都合よく参照できるようにすることができる。

【０００８】

【発明が解決しようとする課題】このタイプのブックマーク・システムを使用可能にすることは、特に小フットプリント・デバイス環境内では望ましい。このようなシステムでは、特定の動作を実行する際の困難さのレベルを著しく低減することができる。これは、ユーザが小フットプリント・デバイスで入力する手段がしばしば非常に限定されるからである。しかし、このタイプのブックマーク・システムの実装は、小フットプリント・デバイス環境にとって特定の難題でもある。これは、所望の抽象化および一般性を達成するためにあるレベルのソフトウェア・インフラストラクチャが必要だからであり、しかもこのソフトウェア・インフラストラクチャが小フットプリント・デバイス上で動作するために十分に軽量でなければならないからである。このタイプのブックマーク・システムを小フットプリント・デバイスに実装するシステムおよび方法を、本明細書に記載する。

【０００９】

【課題を解決するための手段】本発明は、小フットプリント・デバイスのユーザが、様々なタイプのオブジェクトおよび／またはデータ・ソース（本明細書では総称してデータ・ソースと呼ぶ）を参照するブックマークを作成し、格納することができるようにする。各ブックマークは、ユニフォーム・リソース・ロケータ（ＵＲＬ）を含むことができ、これを使用してオブジェクト／データ・ソースを参照することができる。このブックマーク・システムはオープン・エンドにすることができ、実質的にいかなるタイプのオブジェクトまたはデータ・ソースにもブックマークを付けることができる。後にユーザまたはアプリケーションがブックマークを使用してそれぞれのデータ・ソースを参照し、表示や編集など、あるタイプのアクションをそのデータ・ソースに対して実行することができる。

【００１０】たとえば、ユーザは、電子メール・クライアント内から特定の電子メールにブックマークを付けることができる。この場合、ブックマークはあるタイプのカプセル化オブジェクトを参照することができる。これはデータ・ソースと呼ばれ、電子メールを表現するデータを含んでいる。したがって、ブックマーク・サービスは、様々なタイプのエンティティをカプセル化することができる起動フレームワークと協調することができる。起動フレームワークは、１つのタイプを１つのデータソースに割り当てることができ、これによってデータ・ソースを他の種類のデータ・ソースから区別する。起動フレームワークは、特定の「動詞」をデータ・ソース・タイプに関連付けることもでき、特定の実行可能モジュールまたはルーチンをデータ・ソース・タイプおよび動詞に関連付けることもでき、そこでこの実行可能モジュールを呼び出して、この動詞によって特定のデータ・ソースに指示されたアクションを実行することができる。

【００１１】ブックマーク・サービス・モジュールは、小フットプリント・デバイスのアプリケーション／サー

ビス・コンテインメント・フレームワーク内で実行し、他のアプリケーション／サービスによって、特定のデータ・ソースを参照するブックマークを作成して格納するために呼び出すことができる。小フットプリント・デバイス用の軽量のアプリケーション／サービス・フレームワークを、以下に説明する。このブックマーク・サービスは、上述のように、起動フレームワークと共に動作することができる。一実施形態では、ブックマーク・サービスがＪａｖａＢｅａｎｓ™起動フレームワーク（ＪＡＦ）と共に動作して、上述の機能性を実装する。

【００１２】本発明の他の目的および利点は、以下の詳細な説明を読み、添付の図面を参照することによって明らかになるであろう。

【００１３】

【発明の実施の形態】本発明は、様々な修正および代替形式が可能であり、その特定の実施形態を例として図面に示し、本明細書で詳細に説明する。しかし、図面および詳細な説明は、本発明を、開示した特定の形式に限定することを意図するものでなく、逆に、添付の特許請求の範囲によって定義された本発明の精神および範囲内に入るすべての修正、均等物および代替物を包含することを意図するものである。

【００１４】図１－データ・ソースの例

図１は、ブックマークを付けることができるデータ・ソースのいくつかの例である。上述のように、ブックマーク・サービスはオープンな起動フレームワークと共に動作することができ、様々なタイプのデータ・ソースを型付けし、カプセル化することができる。図１は、いくつかの例示的タイプのデータ・ソースであり、小フットプリント・デバイス上で動作する典型的なアプリケーション・プログラムのユーザがブックマークを付けることができるものである。

【００１５】図１のように、ユーザは特定の電子メールにブックマークを付けることができる。たとえば、ユーザは、電子メールを読んでいる間に、その電子メールのブックマークを作成する特定のコマンドを選択することができる。ユーザは後に、このブックマークを使用してこの電子メールを高速で参照することができる。たとえば、ユーザは、ブックマークのリストを表示するブックマーク・サービスを呼び出すことができる。図１は、ブックマークを付けることができる他のデータ・ソースとして、ウェブ・ブラウザ内からブックマークを付けられたウェブ・ページ、および個人情報マネージャ・プログラム内からブックマークを付けられたアポイントメント・エントリなどを示す。

【００１６】ブックマークは様々な方法で編成することができる。たとえば、システムは、すべてのブックマークが保存されている中央リストを有することができ、あるいは別々のアプリケーションがそれ自体のブックマーク・リストを有することができ、あるいはこれらの手法

の様々な組み合わせを取ることができる。ブックマークを作成する要求は、ブックマーク・サービスを呼び出してブックマークへのデータ・ソースを選択したり、あるいはアプリケーション内からブックマーク・サービスを呼び出すコマンドを出すなど、様々な状況で発生する可能性がある。一実施形態では、ブックマークを別のシステムからインポートすることができる。

【００１７】図１のブックマークは、ローカルまたはリモートのデータ・ソースを参照することができる。たとえば、図示のウェブ・ページのブックマークは、リモート・サーバに格納されたＨＴＭＬページを参照することができ、電子メールのブックマークは、別のシステムに格納された電子メールを参照することができる。

【００１８】図２－ブックマークの作成

図２は、特定のデータ・ソースを参照するブックマークを作成するプロセスを示すフローチャートである。図２がブックマーク作成プロセスの一実施形態を示すことに留意されたい。他の実施形態は、様々なステップを追加し、結合し、修正し、あるいは省略することによって可能である。

【００１９】図２のステップ６００で、ユーザがブックマーク・サービスを呼び出す。たとえば、ユーザは、プルダウン・メニュー・コマンドを出してブックマーク・サービスを呼び出すことができ、次いでブックマーク・サービスのグラフィカル・ユーザ・インタフェースをディスプレイに表示することができる。ステップ６０２で、ユーザがブックマークを付ける特定のデータソースを選択する。たとえば、ユーザはディレクトリ構造を走査検索して、仮想名刺、画像など、特定のタイプのデータ・ソースを含むファイルを見つけることができる。ステップ６００およびステップ６０２は、単一のステップに結合することができる。たとえば、ユーザは、アプリケーション内から、現在選択されているデータ・ソースにブックマークを付けるようにブックマーク・サービスに指令するコマンドを出すことができる。

【００２０】ステップ６０４で、ブックマーク・サービスが、ステップ６０２で選択したデータ・ソースを参照するブックマーク・エントリを作成する。ブックマーク・エントリは、ステップ６０２で選択した特定のデータ・ソースを識別する情報を含む。一実施形態では、ブックマーク・エントリがＵＲＬを含む。

【００２１】ステップ６０６で、ブックマーク・サービスが、ステップ６０４で作成したブックマーク・エントリを格納する。たとえば、一実施形態では、ブックマーク・エントリを、このブックマーク・サービス自体からアクセス可能であり、かつシステム内で動作している他のアプリケーションからもアクセス可能であるブックマークの中央リストに、格納することができる。別の実施形態では、システムが、異なるアプリケーションまたはデータ・ソース・タイプ毎に別々のブックマーク・リス

トを保守することができる。

【００２２】図３－ブックマークを付けたデータ・ソースの参照

図３は、ブックマークを使用してそれぞれのデータ・ソースを参照するプロセスを示すフローチャートである。図３がブックマークを参照するプロセスの一実施形態を示すことに留意されたい。他の実施形態は、様々なステップを追加し、結合し、修正し、あるいは省略することによって可能である。

【００２３】ステップ６１０で、ユーザがブックマーク・エントリを選択する。ブックマークの選択は、様々な状況で発生する可能性がある。たとえば、ユーザはブックマーク・サービスを呼び出して、ブックマーク・サービスによってユーザに提示されたリストから特定のブックマーク・エントリを選択することができる。ユーザは、たとえばブックマーク・メニューから特定のメニュー項目を選択することによって、アプリケーション内から特定のブックマークを選択することもできる。

【００２４】ステップ６１２で、ユーザが、ステップ６１０で選択したブックマーク・エントリによって参照されたデータ・ソースに対して実行するアクションを選択する。たとえば、ユーザは、ブックマークを付けた電子メールの「編集」アクションや、ブックマークを付けた画像の「表示」アクションなどを選択することができる。ユーザは、実行するアクションを、可能なアクションのリストから選択することによって選択することができ、あるいは、実行するアクションを、ユーザのアクションのコンテキストから暗示的にすることができる。

【００２５】ステップ６１４で、ステップ６１２で指定したアクションを、ステップ６１０で選択したブックマーク・エントリによって参照されたデータ・ソースに対して実行するために、適切なプログラム・モジュールが呼び出される。ステップ６１２および６１４は、アクションをデータ・ソース・タイプに定義することができ、特定のプログラム・モジュールをこれらのアクションにマッピングすることができる、起動フレームワークが存在することを仮定する。一実施形態では、小フットプリント・デバイス内で動作するブックマーク・サービスおよび／またはアプリケーション・プログラムが、ＪａｖａＢｅａｎｓ™起動フレームワーク（ＪＡＦ）を利用して、ステップ６１２および６１４を実装する。ＪａｖａＢｅａｎｓ™起動フレームワークについては、以下で説明する。

【００２６】ステップ６１０、６１２および６１４は、単一のステップに結合することができる。たとえば、ユーザは、個人用コンタクト・リスト・サービス内から、ブックマークが付いた個人用コンタクト・エントリのリストを表示するメニューにアクセスすることができる。次いで、ユーザは、このメニューからコンタクト・エントリを選択することができ、これによってアプリケーションにコンタクト・エントリを表示させることができる。

【００２７】ＪａｖａＢｅａｎｓ™起動フレームワーク一実施形態では、ＪａｖａＢｅａｎｓ™起動フレームワーク（ＪＡＦ）を利用して、データ・ソースをカプセル化し、データ・ソースにアクションを定義するなどのためのインフラストラクチャを提供する。このセクションでは、ＪａｖａＢｅａｎｓ™起動フレームワークを説明する。

【００２８】ＪＡＦは、いくつかの関連サービスを実装する。これらのサービスには、任意データのタイプを決定すること、データへのアクセスをカプセル化すること、特定のタイプのデータに対して使用可能な操作を発見すること、データの特定部分に対する所望の操作に対応するソフトウェア・コンポーネントをインスタンス化することが含まれる。ＪＡＦは、Ｓｔａｎｄａｒｄ Ｅｘｔｅｎｓｉｏｎとして Ｊａｖａ™プラットフォームにパッケージ化されている。

【００２９】図４は、ＪＡＦアーキテクチャを含む主要な要素である。ここに図示したフレームワークは、特定の応用例に結び付けられるものでないことに留意されたい。

【００３０】図４のＤａｔａＨａｎｄｌｅｒクラスは、ＪＡＦ認識クライアントと他のサブシステムの間の一貫したインタフェースを提供する。

【００３１】ＤａｔａＳｏｕｒｃｅインタフェースは、データを含むオブジェクトであって、データ・アクセスを提供するストリームとこのデータを記述するＭＩＭＥタイプを定義する文字列とを返すことができるオブジェクトをカプセル化する。クラスは、共通のデータ・ソース（ウェブ、ファイル・システム、ＩＭＡＰ、ｆｔｐなど）について実装することができる。ＤａｔａＳｏｕｒｃｅインタフェースは、データ・ソース毎にユーザがカスタマイズできるように拡張することもできる。ＤａｔａＳｏｕｒｃｅをＤａｔａＨａｎｄｌｅｒで設定した後、クライアントがそのデータに対して使用可能な操作を決定することができる。

【００３２】ＪＡＦは、便宜上、次の２つのＤａｔａＳｏｕｒｃｅクラス実装を含む。ＦｉｌｅＤａｔａＳｏｕｒｃｅは、ファイル内で保持されているデータにアクセスする。ＵＲＬＤａｔａＳｏｕｒｃｅは、ＵＲＬで保持されているデータにアクセスする。

【００３３】ＣｏｍｍａｎｄＭａｐは、そのインタフェースのコンシューマが、特定のＭＩＭＥタイプ上で使用可能な「コマンド」、ならびに特定のＭＩＭＥタイプのオブジェクトに対して動作できるオブジェクト（実際上はコンポーネント・レジストリ）を検索するためのインタフェースを決定できるようにするサービスを提供する。Ｃｏｍｍａｎｄ Ｍａｐは、ＣｏｍｍａｎｄＭａｐの特定のインスタンスの実装によって定義された機構に

よって、特定のデータ・タイプに対して使用可能な機能のリストを生成し、保守することができる。

【００３４】ＪａｖａＢｅａｎｓ™パッケージは、コマンドを実装したソフトウェア・コンポーネントのプログラミング・モデルを提供する。各ＪａｖａＢｅａｎｓ™コンポーネントは、客観化（externalization）を使用することができ、あるいは、ＣｏｍｍａｎｄＯｂｊｅｃｔインタフェースを実装して、型付けされたデータをそれに渡すことができるようにすることができる。

【００３５】ＪＡＦはＣｏｍｍａｎｄＭａｐインタフェースを定義し、これは柔軟性があり拡張可能なＣｏｍｍａｎｄＭａｐ用フレームワークを提供する。ＣｏｍｍａｎｄＭａｐインタフェースでは、開発者が、どのコマンドがシステム上で使用可能であるかを発見するための独自のソリューションを開発することができる。可能な実装では、プラットフォーム上の「タイプ・レジストリ」にアクセスでき、あるいはサーバ・ベースのソリューションを使用することができる。ＪＡＦは、機能性のような、ＲＦＣ　１５２４（．ｍａｉｌｃａｐ）に基づく簡素なデフォルト・ソリューションを提供する。

【００３６】Ｂｅａｎｓは、ＪＡＦサービスと対話するためにＣｏｍｍａｎｄＯｂｊｅｃｔインタフェースを拡張する。ＪＡＦを認識するＪａｖａＢｅａｎｓ™コンポーネントは、データ・タイプを検索してそのデータに対して動作するために、それらのＤａｔａＳｏｕｒｃｅおよびＤａｔａＨａｎｄｌｅｒオブジェクトに直接アクセスすることができる。

【００３７】ＪＡＦフレームワークの使用

ＪＡＦフレームワークの「標準」コンシューマは、ＤａｔａＨａｎｄｌｅｒを介してこのフレームワークにアクセスする（ただし、主要なサブシステムは独立して動作するようにも設計されている）。下層ＤａｔａＳｏｕｒｃｅオブジェクトは、ＤａｔａＨａｎｄｌｅｒクラスが構築されたときに、ＤａｔａＨａｎｄｌｅｒに関連付けられる。ＤａｔａＨａｎｄｌｅｒは、データ型付け情報をＤａｔａＳｏｕｒｃｅから検索するか、あるいはデータ・タイプを直接コンストラクタから得る。この初期化ステップが完了すると、このデータ項目に対して実行できるコマンドのリストの要求に、ＤａｔａＨａｎｄｌｅｒからアクセスすることができる。

【００３８】アプリケーションがこのリストの要求を出したとき、ＤａｔａＨａｎｄｌｅｒは、使用可能なコマンドのリストをＣｏｍｍａｎｄＭａｐオブジェクトから要求するために、返されるＭＩＭＥデータ・タイプのスペシファイアー（specifier）を使用する。ＣｏｍｍａｎｄＭａｐは、使用可能なコマンド（Ｂｅａｎｓとして実装される）およびそれらのサポートされているデータ・タイプの知識を有する。ＣｏｍｍａｎｄＭａｐは、要求されたＭＩＭＥタイプおよびＣｏｍｍａｎｄＭａｐ実装の意味に基づくすべてのコマンドの完全リストのサブセットを、ＤａｔａＨａｎｄｌｅｒに返す。

【００３９】最後に、アプリケーションがコマンドをあるデータに適用したいとき、これは適切なＤａｔａＨａｎｄｌｅｒインタフェースを通じて実施される。このインタフェースは、ＣｏｍｍａｎｄＭａｐを使用して、このデータに対して動作するために使用する適切なＢｅａｎを検索する。コンテナ（フレームワークのユーザ）が、データとＢｅａｎの間の関連付けを行う。

【００４０】ＪＡＦ使用シナリオ

このシナリオでは、ＪＡＦを実装しているときに呼び出される標準のタスクの流れを例示するために、仮のファイル・ビューワ・アプリケーションの例を使用する。このファイル・ビューワは、ＣＤＥの「ｄｔｆｉｌｅ」またはＷｉｎｄｏｗｓ（登録商標）　９５　Ｅｘｐｌｏｒｅｒユーティリティに類似している。このアプリケーションを開始したとき、使用可能なファイルの表示がユーザに提示される。これは、ＣＤＥのｄｔｆｉｌｅまたはＥｘｐｌｏｒｅｒの「右マウス」メニューのような機能を含んでおり、選択されたデータ項目に対して実行できるすべての操作が、その項目のポップアップ・メニューにリストされる。

【００４１】一般的なユーザは、このアプリケーションを開始して、ファイルのディレクトリを閲覧する。ユーザがファイルをクリックして指定したとき、このアプリケーションはそのファイルに対して使用可能な操作をリストするポップアップ・メニューを表示する。ファイル・システム・ビューワ・ユーティリティは通常、「編集」、「表示」および「印刷」コマンドを、使用可能な操作として含む。たとえば「表示」を選択すると、このユーティリティは、選択されたファイルを、そのファイルに保持されているデータ・タイプのデータを表示できるビューワ内で開く。

【００４２】このアプリケーションによって実行されるタスクの説明を、明瞭にするために次の３つの離散ステップに分類する。

【００４３】初期化、すなわち、アプリケーションがファイル・システムの表示を構築する。

【００４４】コマンド・リストを得る、すなわち、アプリケーションが、選択されたデータ項目についてのコマンド・リストを提示する。

【００４５】コマンドを実行する、すなわち、アプリケーションが、選択されたデータ・オブジェクトに対してコマンドを実行する。

【００４６】初期化

以下に述べるインターフェイスの１つは、「ＤａｔａＳｏｕｒｃｅ」オブジェクトである。ＤａｔａＳｏｕｒｃｅオブジェクトは、下層データ・オブジェクトをクラス内にカプセル化する。このクラスは、下層データ格納機構を抽象化し、そのコンシューマに共通のデータ・アクセスおよび型付けインタフェースを提示するものであ

る。ファイル・ビューワ・アプリケーションは、ファイル・システムの内容を照会する。

【００４７】ビューワは、ディレクトリ内の各ファイルについてＤａｔａＳｏｕｒｃｅオブジェクトをインスタンス化する。次いでビューワは、ＤａｔａＳｏｕｒｃｅをそのコンストラクタの引数として、ＤａｔａＨａｎｄｌｅｒをインスタンス化する。ＤａｔａＨａｎｄｌｅｒは、ＤａｔａＳｏｕｒｃｅを使わずにインスタンス化することができない。ＤａｔａＨａｎｄｌｅｒオブジェクトは、クライアント・アプリケーションにＣｏｍｍａｎｄＭａｐへのアクセスを提供し、これによって、そのデータに対して動作できるコマンドへのアクセスを可能にするサービスを提供する。このアプリケーションは、ＤａｔａＨａｎｄｌｅｒオブジェクトのリストを保守し、それらの名前およびアイコンを照会してその表示を生成する。

```
// ディレクトリ内の各ファイルに対して
File file = new File(file_name);
DataSource ds = new FileDataSource(file);
DataHandler dh = new DataHandler(ds);
```

【００４８】コマンド・リストを得る

アプリケーションが初期化され、ファイルのリストがユーザに提示された後、ユーザはこのリストでファイルを選択することができる。ユーザがファイルを選択したとき、アプリケーションは、そのファイルに対して使用可能な操作をリストするポップアップ・メニューを表示する。

【００４９】アプリケーションは、ファイルに関連付けられたＤａｔａＨａｎｄｌｅｒオブジェクトから使用可能なコマンドのリストを要求することによって、この機能性を実装する。ＤａｔａＨａｎｄｌｅｒは、ＤａｔａＳｏｕｒｃｅオブジェクトからデータのＭＩＭＥタイプを検索し、そのタイプに対して使用可能な操作をＣｏｍｍａｎｄＭａｐに照会する。アプリケーションはこのリストを解釈し、これをユーザにポップアップ・メニューで提示する。次いで、ユーザがこのリストから操作を１つ選択する。

```
// オブジェクトのコマンド・リストを得る
CommandInfo cmdInfo[] = dh.getPreferredCommands();

PopupMenu popup = new PopupMenu("Item Menu");


// ポップアップに使用可能なコマンドを入れる
for(i = 0; i < cmdInfo.length; i++)
    popup.add(cmdInfo[i].getCommandName());

// ポップアップを追加して表示する
add(popup);
popup.show(x_pos, y_pos);
```

【００５０】コマンドを実行する

ユーザがポップアップ・メニューからコマンドを選択した後、アプリケーションは適切なＣｏｍｍａｎｄＩｎｆｏクラスを使用して、選択されたコマンドに対応するＢｅａｎを検索し、適切な機構（ＤａｔａＨａｎｄｌｅｒ、Ｅｘｔｅｒｎａｌｉｚａｔｉｏｎなど）を使用してデータをこのＢｅａｎに関連付ける。いくつかのＣｏｍｍａｎｄＯｂｊｅｃｔ（たとえばビューワ）はｊａｖａ．ａｗｔ．Ｃｏｍｐｏｎｅｎｔからサブクラス化され、それらが親コンテナを与えられる必要がある。他のＣｏｍｍａｎｄＯｂｊｅｃｔ（デフォルトのｐｒｉｎｔＣｏｍｍａｎｄなど）は、ユーザ・インタフェースを提示しない可能性がある。これによって、スタンド・アロンのビューワ／エディタ、あるいは複合ドキュメント・システム内のコンポーネントとして機能するために十分な柔軟性をそれらに持たせることができる。「アプリケーション」は、ＣｏｍｍａｎｄＯｂｊｅｃｔが実行するために適当な環境（コンテインメント、ライフ・サイクルなど）を提供する責任を有する。

```
// コマンド・オブジェクトを得る
Object cmdBean = cmdInfo[cmd_id].getCommandObject(dh,
                                    this.getClassLoader());
... // 適切な場合に直列化／客観化を使用する
my_awt_container.add((Component)cmdBean);
```

【００５１】代替シナリオ

第１のシナリオは「標準」の場合であった。アプリケーションがすでにそのデータを表現するオブジェクトを作成済みである状況もある。この場合、既存のオブジェクトをＩｎｐｕｔＳｔｒｅａｍに変換したＤａｔａＳｏｕｒｃｅのイン・メモリ・インスタンスを作成することは、システム・リソースを非効率的に使用することであり、データの忠実性が損なわれる結果となる可能性がある。

【００５２】このような場合、アプリケーションは、ＤａｔａＨａｎｄｌｅｒ（Ｏｂｊｅｃｔ　ｏｂｊ、Ｓｔｒｉｎｇ　ｍｉｍｅＴｙｐｅ）コンストラクタを使用して、ＤａｔａＨａｎｄｌｅｒをインスタンス化することができる。ＤａｔａＨａｎｄｌｅｒは、Ｔｒａｎｓｆｅｒａｂｌｅインタフェースを実装するので、消費側のＢｅａｎがＩｎｐｕｔＳｔｒｅａｍ以外の表現を要求することができる。ＤａｔａＨａｎｄｌｅｒは、要求するコンシューマ用にＤａｔａＳｏｕｒｃｅも構築する。ＤａｔａＣｏｎｔｅｎｔＨａｎｄｌｅｒ機構は、ＯｂｊｅｃｔからＩｎｐｕｔＳｔｒｅａｍへの変換もできるように

拡張される。

【００５３】以下のコードは、ＪＡＦを使用したデータ・ベース・フロント・エンドの一例であり、オブジェクトに関する照会結果を提供するものである。

```
/**
*  ビューワに自分の照会結果を表示させる
*/
Component getQueryViewer(QueryObject qo) throws Exception {
    String mime_type = qo.getType();
    Object q_result = qo.getResultObject();
    DataHandler my_dh = new DataHandler(q_result, mime_type);

    return (Component)my_dh.getCommand("view").getCommandObject(my_dh,
null));
}
```

【００５４】ＪＡＦフレームワーク・コア・クラス

インタフェースＤａｔａＳｏｕｒｃｅ：ＤａｔａＳｏｕｒｃｅインタフェースは、ＪａｖａＢｅａｎｓ™起動フレームワークに、ある任意のデータ集合の抽象化を提供する。このインタフェースは、そのデータのタイプならびにそのデータへのアクセスを、ＩｎｐｕｔＳｔｒｅａｍｓおよびＯｕｔｐｕｔＳｔｒｅａｍｓの形式で適切な場合に提供する。

【００５５】クラスＤａｔａＨａｎｄｌｅｒ：ＤａｔａＨａｎｄｌｅｒクラスは、多数の異なるソースおよび形式で使用可能なデータへの一貫したインタフェースを提供する。このクラスは、ＤａｔａＣｏｎｔｅｎｔＨａｎｄｌｅｒｓを使用して、文字列変換および関連する操作への単純なストリームを管理する。このクラスは、そのデータに対して動作できるコマンドへのアクセスを提供する。これらのコマンドは、ＣｏｍｍａｎｄＭａｐを使用して見つけられる。

【００５６】インタフェースＤａｔａＣｏｎｔｅｎｔＨａｎｄｌｅｒ：ＤａｔａＣｏｎｔｅｎｔＨａｎｄｌｅｒインタフェースは、ＤａｔａＨａｎｄｌｅｒによるＴｒａｎｓｆｅｒａｂｌｅインタフェースの実装を機能拡張するために使用できるオブジェクトによって、実装される。ＤａｔａＣｏｎｔｅｎｔＨａｎｄｌｅｒｓを通じて、フレームワークを、ストリームをオブジェクトに変換し、オブジェクトをストリームに書き込むように、拡張することができる。

【００５７】インタフェースＤａｔａＣｏｎｔｅｎｔＨａｎｄｌｅｒＦａｃｔｏｒｙ：このインタフェースは、ＤａｔａＣｏｎｔｅｎｔＨａｎｄｌｅｒｓのファクトリを定義する。このインタフェースの実装は、ＭＩＭＥタイプをＤａｔａＣｏｎｔｅｎｔＨａｎｄｌｅｒのインスタンスにマッピングするべきである。このインタフェースを実装するクラスの設計パターンは、ｊａｖａ．ｎｅｔ．ＵＲＬで使用するＣｏｎｔｅｎｔＨａｎｄｌｅｒ機構の設計パターンと同じものである。

【００５８】クラスＣｏｍｍａｎｄＭａｐ：ＣｏｍｍａｎｄＭａｐクラスは、システム内で使用可能なビューワ、エディタ、印刷などのオブジェクトのレジストリへのインタフェースを提供する。開発者は、このパッケージに含まれているＣｏｍｍａｎｄＭａｐ実装（ＭａｉｌｃａｐＣｏｍｍａｎｄＭａｐ）を使用するか、あるいは独自のものを開発するかのいずれかを行うことが求められる。このクラスのいくつかのメソッドは抽象であることに留意されたい。

【００５９】インタフェースＣｏｍｍａｎｄＯｂｊｅｃｔ：起動フレームワークを認識するＢｅａｎｓは、このインタフェースを実装して、どのコマンド動詞を実行するように求められているかを見出し、操作すべきデータを表現するＤａｔａＨａｎｄｌｅｒを得る。このインタフェースを実装しないＢｅａｎｓも使用することができる。このようなコマンドは、Ｅｘｔｅｒｎａｌｉｚａｂｌｅインタフェースを使用するか、あるいはアプリケーション固有のメソッドを使用して、データを得ることができる。

【００６０】クラスＣｏｍｍａｎｄＩｎｆｏ：ＣｏｍｍａｎｄＩｎｆｏクラスは、ＣｏｍｍａｎｄＭａｐの実装によって、コマンド要求の結果を記述するために使用される。このクラスは、要求側に、要求された動詞ならびにｂｅａｎのインスタンスを提供する。このコマンドを実装するクラス名を返すメソッドもあるが、それが有効な値を返すことは保証されていない。この理由は、ＣｏｍｍａｎｄＩｎｆｏをサブクラス化するＣｏｍｍａｎｄＭａｐの実装が、特殊な挙動を提供できるようにするためである。たとえば、Ｆｒａｍｅｗｏｒｋ　Ｄｅｌｉｖｅｒａｂｌｅｓである。

【００６１】ＪＡＦフレームワーク補助クラス

クラスＦｉｌｅＤａｔａＳｏｕｒｃｅ：ＦｉｌｅＤａｔａＳｏｕｒｃｅクラスは、ファイルをカプセル化する単純なＤａｔａＳｏｕｒｃｅオブジェクトを実装する。このクラスは、ＦｉｌｅＴｙｐｅＭａｐオブジェクトを介してデータ型付けサービスを提供する。

【００６２】クラスＦｉｌｅＴｙｐｅＭａｐ：ＦｉｌｅＴｙｐｅＭａｐは、ファイルについてのデータ型付けインタフェースを提供する抽象クラスである。このクラス

15

の実装はｇｅｔＣｏｎｔｅｎｔＴｙｐｅメソッドを実装し、これがコンテント・タイプをファイル名またはＦｉｌｅオブジェクトから導出する。ＦｉｌｅＴｙｐｅＭａｐは、データ・タイプを決定するために、ファイルのファイル拡張子を検査すること（ＭｉｍｅｔｙｐｅｓＦｉｌｅＴｙｐｅＭａｐなど）から、ファイルを開いてそのタイプをファイルの内容から導出しようと試みることまで、いかなるスキームも使用することができる。ＦｉｌｅＤａｔａＳｏｕｒｃｅクラスは、デフォルトのＦｉｌｅＴｙｐｅＭａｐ（変更されない限りＭｉｍｅｔｙｐｅｓＦｉｌｅＴｙｐｅＭａｐ）を使用して、ファイルのコンテント・タイプを決定する。

【００６３】クラスＭｉｍｅｔｙｐｅｓＦｉｌｅＴｙｐｅＭａｐ：このクラスはＦｉｌｅＴｙｐｅＭａｐを拡張し、ファイルのデータ型付けを、それらのファイル拡張子を介して提供する。このクラスは、．ｍｉｍｅ．ｔｙｐｅｓ形式を使用する。

【００６４】クラス　ＵＲＬＤａｔａＳｏｕｒｃｅ：ＵＲＬＤａｔａＳｏｕｒｃｅクラスは、ＤａｔａＳｏｕｒｃｅインタフェースでＵＲＬオブジェクトをラップするオブジェクトを提供する。ＵＲＬＤａｔａＳｏｕｒｃｅは、ＪａｖａＢｅａｎｓ™起動フレームワーク内でＵＲＬによって記述されたデータの処理を簡単にする。これは、このクラスを使用して新規のＤａｔａＨａｎｄｌｅｒを作成することができるからである。

【００６５】クラスＭａｉｌｃａｐＣｏｍｍａｎｄＭａｐ：ＭａｉｌｃａｐＣｏｍｍａｎｄＭａｐは、ＣｏｍｍａｎｄＭａｐ抽象クラスを拡張する。このクラスは、その構成がｍａｉｌｃａｐファイル（ＲＦＣ１５２４）に基づいているＣｏｍｍａｎｄＭａｐを実装する。ＭａｉｌｃａｐＣｏｍｍａｎｄＭａｐは、プログラムによっても構成ファイルを介しても構成することができる。

【００６６】クラスＡｃｔｉｖａｔｉｏｎＤａｔａＦｌａｖｏｒ：ＡｃｔｉｖａｔｉｏｎＤａｔａＦｌａｖｏｒは、ｊａｖａ．ａｗｔ．ｄａｔａｔｒａｎｓｆｅｒ．ＤａｔａＦｌａｖｏｒの特殊なサブクラスである。このクラスによって、ＪＡＦで、ＤａｔａＦｌａｖｏｒクラスによって新規コンストラクタを介して格納された３つの値すべてを設定できるほか、ｅｑｕａｌｓメソッドでは改良されたＭＩＭＥ解析が可能である。この改良された解析を除いて、この意味はＪＤＫのＤａｔａＦｌａｖｏｒクラスの意味と同じである。

【００６７】クラスＵｎｓｕｐｐｏｒｔｅｄＤａｔａＴｙｐｅＥｘｃｅｐｔｉｏｎ：要求された操作が、要求されたデータ・タイプをサポートしていないことを知らせる。

【００６８】クラスＭｉｍｅＴｙｐｅ：ＲＦＣ２０４５および２０４６で定義されている、Ｍｕｌｔｉｐｕｒｐｏｓｅ　Ｉｎｔｅｒｎｅｔ　Ｅｘｔｅｎｓｉｏｎ（ＭＩＭＥ）タイプ。

16

【００６９】小フットプリント・デバイス・アプリケーション／サービス・コンテインメント・フレームワーク

上述のように、ブックマーク・サービス、または、個人用コンタクト・リスト・サービスなど小フットプリント・デバイス上で実行する別のアプリケーション／サービスは、別々のサービス・モジュールを呼び出して、ユーザが特定のデータ・ソースに選択したアクションを実行することができる。システムをオープン・エンドにすることができ、様々なアクションを様々なタイプのデータ・ソースに定義することができるので、小フットプリント・デバイス・ソフトウェア・アプリケーション／サービスが、モジュール方式で拡張可能な軽量のアプリケーション／サービス・コンテインメント・フレームワークに基づくことが必要であるか、あるいは望ましいことがある。このようなコンテインメント・フレームワークについて、以下に記載する。

【００７０】図５－ハードウェア・アーキテクチャのブロック図

図５は、典型的な小フットプリント・デバイスのハードウェア・アーキテクチャを示すブロック図である。本明細書で使用しているように、小フットプリント・デバイスは、プロセッサおよびシステム・メモリなどのコンピューティング・リソースを含むハードウェア・デバイスであるが、これらのリソースの１つまたは複数に、典型的なデスクトップ・コンピュータにある制約よりも著しく大きい制約がある。たとえば、小フットプリント・デバイスは２メガバイト以下のメモリを有することができるが、典型的なデスクトップ・システムは６４メガバイト以上のメモリを有することができる。また、典型的な小フットプリント・デバイスは、プロセッサのタイプまたはプロセッサの速度、あるいはその両方のいずれかに関して、典型的なデスクトップ・コンピューティング・システムよりも著しく処理能力が劣るであろう。たとえば、携帯情報端末デバイスは１６ＭＨｚのプロセッサを有することができるが、典型的なデスクトップ・システムは１００ＭＨｚ以上のプロセッサ速度を有することができる。また、典型的な小フットプリント・デバイスは、デスクトップ・コンピューティング・システムのディスプレイ画面よりも著しく小さいディスプレイ・サイズを有する。たとえば、ハンドヘルド・コンピュータのディスプレイ画面は、一般にデスクトップ・モニタのディスプレイ画面に比較して小さい。

【００７１】ここに挙げた特定の数値は例示的なものにすぎず、比較のために使用するものであることに留意されたい。たとえば、８メガバイト以上のメモリを有する携帯情報端末は、なお小フットプリント・デバイスである可能性があるが、このデバイスは、上述の典型的な数字である２メガバイトより多いメモリを有している。

【００７２】小フットプリント・デバイスは、上述のメモリ、プロセッサおよびディスプレイ・サイズのリソー

スのほかにも、典型的なデスクトップ・コンピューティング・システムに比較して、他のリソースのタイプに制約があることがある。たとえば、典型的な小フットプリント・デバイスはハード・ディスクを有していないことがあったり、ネットワーク接続を有していないことがあったり、あるいは断続的なネットワーク接続を有したり、あるいは無線ネットワーク接続を有したりすることがある。

【００７３】多数の小フットプリント・デバイスは携帯用であり、かつ／またはデスクトップ・コンピュータに比較して小型であるが、必ずしもそうではない。また、多数の小フットプリント・デバイスは、主として、あるいは専ら、バッテリ駆動型でもある。また、小フットプリント・デバイスは典型的に、典型的なデスクトップ・コンピューティング・システムよりも制限された範囲または狭い範囲の使用可能性を有することもある。小フットプリント・デバイスには、これだけに限定されるものではないが、以下の例が含まれる。すなわち、ハンドヘルド・コンピュータ、ウェアラブル・デバイス（たとえば、腕時計コンピュータ）、携帯情報端末（ＰＤＡ）、「高性能」携帯電話、セット・トップ・ボックス、ゲーム・コンソール、全世界測位システム（ＧＰＳ）ユニット、電子テキストブック・デバイスなどである。新しいクラスのコンシューマ・デバイスが急速に出現しているので、小フットプリント・デバイスの網羅的なリストを提供することは不可能である。しかし、「小フットプリント・デバイス」という用語は、上述の用語の精神および範囲内に適度に含めることができるようなデバイスを含むことを意図するものである。

【００７４】図５は、典型的な小フットプリント・デバイスのブロック図である。小フットプリント・デバイスが、所望の様々な異なるアーキテクチャを有することができることに留意されたい。本発明の動作を理解する上で必要でないハードウェア要素は、簡素にするため省略した。

【００７５】図５のように、小フットプリント・デバイスはプロセッサ１００を含む。プロセッサ１００は、ｘ８６プロセッサを含む様々なタイプのいずれにもすることができる。たとえば、Ｐｅｎｔｉｕｍ（登録商標）クラス、ＰｏｗｅｒＰＣ（登録商標）プロセッサ、ならびに他の低パワーのプロセッサ、または小フットプリント・デバイス専用に開発されたプロセッサである。プロセッサ１００は、デスクトップ・コンピュータ・クラスのプロセッサで見られるクロック速度に類似したクロック速度、ならびに１６ＭＨｚなどの低速を含む、様々なクロック速度を有することができる。

【００７６】図５のように、このデバイスはシステム・メモリ１０２を含む。システム・メモリ１０２は、ＲＡＭまたはＲＯＭを含む様々なタイプのメモリを含むことができる。典型的な小フットプリント・デバイスは、典型的なデスクトップ・コンピュータ・システムに比較して、非常に小さいメモリ記憶容量を有することがある。

【００７７】小フットプリント・デバイスは、１つまたは複数の入力機構も含むことができる。入力機構１０４を図５に示す。入力機構１０４は、特定のデバイスに適した様々なタイプのいずれにもすることができる。たとえば、入力機構は、キーパッド、マウス、トラックボール、タッチ・ペン、マイクロフォンなどにすることができる。

【００７８】小フットプリント・デバイスは、１つまたは複数のディスプレイ機構も含むことができる。ディスプレイ１０６を図５に示す。しかし、小フットプリント・デバイスはディスプレイを含まないことがあり、あるいはオーディオ・スピーカなど別のタイプの出力機構を含むことがある。ディスプレイ機構１０６は、特定のデバイスに適した様々なタイプのいずれにもすることができる。高性能携帯電話などの典型的な小フットプリント・デバイス用のディスプレイ機構は、デスクトップ・コンピュータ・システムのディスプレイに比較して小型であることがある。

【００７９】図６－ハードウェア／ソフトウェアの階層図

図６は、コンテインメント・フレームワーク内でアプリケーションおよびサービスを実行するシステムに含まれているハードウェア／ソフトウェア層の典型的な階層である。この図は例示的なものであり、特定のデバイスまたは実装に適するように、様々な層を追加し、結合し、あるいは省略することができる。

【００８０】図６の基本層がデバイス・ハードウェア層１２０であり、プロセッサおよびシステム・メモリなど、ソフトウェア・システムをサポートするために必要なハードウェア・リソースを含む。一実施形態では、図５の小フットプリント・デバイス・ハードウェアの例など、小フットプリント・デバイスのハードウェアが、図６のハードウェア層１２０を実装する。しかし、他の実施形態では、ハードウェア層１２０は、他のタイプのデバイスで実装することができる。たとえば、スマート・カードなど、典型的な小フットプリント・デバイスよりもさらに大きいリソース上の制約を有するデバイスなどである。

【００８１】図６のように、ハードウェア層の次の上層がオペレーティング・システム層１２２である。当業者には周知であるように、オペレーティング・システムは、デバイス・ハードウェアと、そのデバイス上で動作するソフトウェアの間のインタフェース層として機能し、入出力、メモリ管理などの低レベルのタスクのマネージャとしての機能を果たす。図６のオペレーティング・システム１２２は、図６のより高い層をサポートするいかなる特定のオペレーティング・システムにもすることができる。オペレーティング・システム１２２は、小

フットプリント・デバイスで使用するために適しているか、あるいはそのために特に書かれた、小型で効率的なものにすることができる。たとえば、オペレーティング・システム１２２は、Ｓｕｎ Ｍｉｃｒｏｓｙｓｔｅｍｓ社から入手可能なＪａｖａＯＳオペレーティング・システムにすることができる。

【００８２】一実施形態では、コンテインメント・フレームワークが、Ｊａｖａ™アプリケーション環境において１つまたは複数のＪａｖａ™クラスとして実装される。図６のように、Ｊａｖａ™仮想マシン層１２４およびＪａｖａ™アプリケーション・プログラミング・インタフェース（ＡＰＩ）クラス・ライブラリ層１２６は、オペレーティング・システムの次の上層である。これらの２層が共にＪａｖａ™アプリケーション環境またはＪａｖａ™プラットフォームを作成する。コンテインメント・フレームワークを実装するクラスは、Ｊａｖａ™ライブラリ１２６を使用して構築することができ、バイトコードにコンパイルすることができる。このバイトコードは、Ｊａｖａ™仮想マシン１２４上で実行する命令であり、オペレーティング・システム１２２および／またはデバイス・ハードウェア１２０と対話する。

【００８３】一実施形態では、コンテインメント・フレームワークがＰｅｒｓｏｎａｌＪａｖａ Ｊａｖａ™アプリケーション環境で実装される。これは、高スケーラブルでモジュール方式、かつ構成可能に設計されたＪａｖａ™プラットフォームであり、最低限のシステム・リソースを必要とするものである。ＰｅｒｓｏｎａｌＪａｖａ™は、Ｊａｖａ™仮想マシンおよびＪａｖａ™ ＡＰＩのサブセットを含み、コアおよびオプションのＡＰＩおよびクラス・ライブラリを含む。さらに、ＰｅｒｓｏｎａｌＪａｖａ™ ＡＰＩは、専用バージョンのＪａｖａ™ ａｂｓｔｒａｃｔ ｗｉｎｄｏｗ ｔｏｏｌｋｉｔ（ＡＷＴ）など、リソースが制限された環境でコンシューマ・アプリケーションが要求する特定の機能を含む。ＰｅｒｓｏｎａｌＪａｖａ™ ＡＷＴライブラリは、コンシューマ製品のルック・アンド・フィールを目標として調整されており、グラフィックスおよびウィンドウ機能を提供し、低解像度のディスプレイおよび代替入力デバイスをサポートする（マウスおよびキーボードのないデバイス用に拡張されたイベント・モデルを介する）。

【００８４】再度図６を参照して、コンテインメント・フレームワーク１２８をＪａｖａ™プラットフォーム層の次の上層として示す。上述のように、コンテインメント・フレームワーク１２８を他のプラットフォームに基づかせることもできる。以下で詳細に述べるように、コンテインメント・フレームワーク１２８は、たとえばモジュール登録、ルックアップ、インスタンス追跡などを可能にすることによって、プログラム・モジュールを管理する。モジュールは様々なサービスを提供することができる。コンテインメント・フレームワーク１２８は、モジュールが他のモジュールを要求してそれらのサービスを使用できるようにする。アプリケーションは、他のモジュールのサービスを利用するモジュールとして実装することができる。したがって、コンテインメント・フレームワーク１２８は、軽量で拡張可能なサービスおよびアプリケーション・フレームワークを提供し、アプリケーションが共存してモジュラー・コード・ベースを共有できるようにする。

【００８５】このタイプの拡張可能なアーキテクチャは、多数のプログラム・モジュールを協調可能にするものであり、小フットプリント・デバイスには重要な開発である。小フットプリント・デバイスは、歴史的に、相対的に狭い用途に制限されてきた。たとえば、携帯電話は通常、テレフォニーおよび他のわずかな用途に使用されていた。しかし、様々な技術が開発され、小フットプリント・デバイスが「より高性能」になり、汎用プロセッサ、より大型のディスプレイ画面などを有することが可能になるので、小フットプリント・デバイスで使用されるアプリケーションの範囲を拡張することが望ましくなっている。

【００８６】このコンテインメント・フレームワークは、一般にデスクトップ・コンピューティング環境に関連するタイプのアプリケーションおよびサービスが、小フットプリント・デバイスにおいて、デスクトップ・コンピュータのユーザが慣れている様式で、共に動作できるようにすることができる。図６および上述のように、小フットプリント・デバイス上で動作するサービスおよびアプリケーション１３０は、コンテインメント・フレームワーク層１２８上で構築されたモジュールとして実装することができる。たとえば、Ｓｕｎ Ｍｉｃｒｏｓｙｓｔｅｍｓ社から入手可能なＰｅｒｓｏｎａｌ Ａｐｐｌｉｃａｔｉｏｎスーツは、コンテインメント・フレームワーク１２８の一実施形態を用いて構築されている。Ｐｅｒｓｏｎａｌ Ａｐｐｌｉｃａｔｉｏｎ Ｓｕｉｔｅは、ブラウザ、電子メール・クライアント、およびパーソナル・オーガナイザなどのアプリケーションの統合セットを含んでいる。

【００８７】図６は、コンテインメント・フレームワーク１２８のいくつかの実施形態の、オフ・デバイス・サービス１３２をオン・デバイス・アプリケーション／サービス１３０と統合する能力も示す。たとえば、コンテインメント・フレームワーク１２８は、小フットプリント・デバイスとＪｉｎｉネットワークなどのネットワークの間のインタフェースを提供することができる。小フットプリント・デバイス・システムは、そのサービスを、ネットワーク内の他のデバイスまたはクライアントが使用するように登録することができる。コンテインメント・フレームワークは、小フットプリント・デバイス内のサービスおよびアプリケーションが、他のネットワ

ーク・デバイスが提供するサービスをルック・アップおよび使用できるようにすることもできる。小フットプリント・デバイスのサービスをネットワーク・サービスと統合することについては、以下で図７についてより詳細に論ずる。

【００８８】図７－１１：例示的ネットワーク・デバイスおよびサービス・フェデレーション

図７は例示的ネットワークであり、コンテインメント・フレームワーク内のアプリケーション／サービスを実行する小フットプリント・デバイスが、ローカル・サービス・ベースのネットワークに接続される。図示の例では、コンテインメント・フレームワーク１４４を利用する高性能携帯電話１３４が、ネットワークに接続される。図ではまた、ネットワークに、プリンタ１３０およびインターネット使用可能テレビ１３２も接続されている。この例では、プリンタ１３０およびテレビ１３２のデバイスが、サービスをネットワークにエクスポートし、場合によってはネットワーク上の他のデバイスのサービスを使用するように、動作可能であると仮定する。たとえば、プリンタがそのプリント・サービス１３８をエクスポートすることができ、インターネット・テレビがそのプリント・サービスをルックアップして使用し、ウェブ・ページを印刷することができる。デバイスおよびサービスのフェデレーションをこの方法で容易にするために、ルックアップ・サービス１３６はネットワーク上に位置している。ルックアップ・サービス１３６は、ネットワーク・サーバなどの別のデバイス上に存在することができる。

【００８９】デバイスおよびサービスのフェデレーションは、様々な方法で実装することができる。たとえば、Ｓｕｎ Ｍｉｃｒｏｓｙｓｔｅｍｓ社から入手可能なＪｉｎｉ™技術は、図７のタイプの分散システムを使用可能にするコンポーネントおよびプログラミング・モデルを含む。一実施形態では、図７のローカル・ネットワークをＪｉｎｉ™ネットワークにすることができ、プリンタ１３０およびインターネット・テレビ１３２をＪｉｎｉ™使用可能デバイスにすることができる。各デバイスは、Ｊｉｎｉ™ネットワーク・ルックアップ・サービスを見つけて、そのデバイスが提供するサービスをこのルックアップ・サービスに登録するように、動作可能である。このルックアップ・サービスは、サービスが提供する機能性を示すインタフェースを、そのサービスを実装するオブジェクトのセットにマッピングする。

【００９０】デバイスまたは他のサービス・プロバイダは、そのサービスをサービス・フェデレーションへ追加するために、まず「発見」プロトコルを使用して適切なルックアップ・サービスを探し出すことができる。図８は、発見プロセスである。図示のように、サービス・プロバイダ１６４、たとえば図７のプリンタ１３０は、ローカル・ネットワーク上で、ルックアップ・サービスにそれら自体を識別させる要求を同報通信することができる。

【００９１】サービス・プロバイダ１６４がルックアップ・サービス１６０を探し出した後、サービス・プロバイダ１６４は、「結合」プロトコルを使用して、そのサービスをルックアップ・サービス１６０に登録することができる。図９は、結合プロセスである。サービス・プロバイダ１６４は、クライアントがそのサービスを呼び出すために使用できるサービス・オブジェクトを作成することができる。図９のように、次いで、提供されたサービスのサービス・オブジェクトを、提供されたサービスのタイプや名前についての情報を含むサービス属性または記述子と共に、ルックアップ・サービス１６０にロードすることができる。たとえば、Ｊｉｎｉ™ネットワークでは、図７のプリンタ１３０が、プリント・サービス１３８用のＪａｖａ™プログラミング・インタフェースを含むサービス・オブジェクトを作成することができる。次いで、プリンタ１３０は、ルックアップ・サービス１３６の「登録」メソッドを呼び出して、このサービス・オブジェクトを、登録中のサービス１３８がプリント・サービス、印刷解像度、可能な用紙サイズなどであると指定する属性と共に、渡すことができる。

【００９２】サービス・プロバイダ１６４がそのサービスをルックアップ・サービス１６０に結合した後、他のネットワーク・クライアントがこのサービスを要求し、使用することができる。サービスを要求するプロセスはルックアップと呼ばれ、これを図１０に示す。ルックアップ・サービスを発見した後、クライアント１６２は、要求するサービスの記述を使用して、ルックアップ・サービス１６０からサービスを要求することができる。ルックアップ・サービス１６０は、要求側から与えられた記述を、ルックアップ・サービスに結合されたサービスに突き合わせようと試みる。ルックアップ・サービス１６０は、結合プロセス中にサービス・プロバイダ１６４から送信されたサービス属性を使用して、この突合せを実行することができる。合致が発見された場合、ルックアップ・サービス１６０は適切なサービス・オブジェクトをクライアント１６２へ提供する。たとえば、要求されたサービス用のＪａｖａ™インタフェースを、クライアント１６２へ提供することができる。

【００９３】クライアント１６２がサービス・オブジェクトをルックアップ・サービスから受け取った後、クライアントがこのサービスを呼び出すことができる。図１１は、サービスを呼び出すプロセスである。サービスが呼び出されたとき、クライアント１６２およびサービス・プロバイダ１６４は直接互いに通信することができる。様々な対話プロトコルのいずれも、この通信に使用することができる。たとえば、使用するプロトコルを、Ｊａｖａ™ Ｒｅｍｏｔｅ Ｍｅｔｈｏｄ Ｉｎｖｏｃａｔｉｏｎ（ＲＭＩ）、ＣＯＲＢＡ、ＤＣＯＭなどにす

23

ることができる。クライアントがルックアップ・サービスから受け取るサービス・オブジェクトは、たとえばＲＭＩメソッドを呼び出すことによって、サービス・プロバイダに位置するコードへコール・バックすることができ、あるいは、要求されたサービスを提供するようにローカルで実行することができ、あるいは、これらの手法を組み合わせて使用することができる。

【００９４】図７のように、ローカル・ネットワーク用のルックアップ・サービス１３６は、インターネット１５４など外部ネットワークへのゲートウェイとしても動作することができる。したがって、サービス・ベースの分散コンピューティング・モデルを、ローカル・ネットワーク外部に位置するクライアントおよびサービスを含めるように拡張することができる。たとえば、Ｏｐｅｎ Ｓｅｒｖｉｃｅ Ｇａｔｅｗａｙ Ｉｎｉｔｉａｔｉｖｅ（ＯＳＧＩ）用に開発されている技術を利用して、このタイプの分散コンピューティング・システムを実装することができる。

【００９５】このタイプの、異なるネットワークとインターネットの間のサービス共有によって、新しいタイプのアプリケーションを開発可能にすることができる。たとえば、商人はインターネット・サービスを使用して特定の顧客についてのデータを記録することができ、広告サービス・プロバイダはこのデータを使用して、どのローカル・ネットワークにコンシューマ・デバイスが接続されているかなどに応じて、コンテキスト特異的な広告をコンシューマ・デバイスに出すことができる。たとえば、顧客はショッピング・モールに入り、無線接続を介して携帯情報端末（ＰＤＡ）をこのショッピング・モールのローカル・ネットワークに接続することができる。インターネット・ベースの顧客データ・サービスをショッピング・モール・ネットワーク用のルックアップ・サービスに結合することができ、このモール・ネットワークに接続したばかりの特定の顧客についての情報を提供することができる。次いで、このショッピング・モール・ネットワーク内で動作するサービスは、このデータを、モール内の顧客の現在位置、時間などの他の要素と共に使用して、個別化された広告を生成して顧客のＰＤＡに出すことができる。

【００９６】図７のネットワークに基づく他の多数のサービス例が可能である。たとえば、住宅内のネットワーク使用可能コンシューマ・デバイスは、電力会社が提供する、住宅内の電力消費量を管理するサービスを、インターネットを介して利用することができ、セキュリティ・サービス・プロバイダは、ネットワーク・サービスを介して家庭または特定のデバイスを監視することができ、資産が荒らされたときに直ちに所有者に通知することができ、健康サービス・プロバイダは、医療機器と通信することによって患者の容態をリモートで監視することができる。

24

【００９７】上に挙げた例では、デバイスが透過的にネットワークに接続することができ、ネットワーク・サービスをデバイス常駐サービスと統合することができ、ネットワーク・クライアントが使用するためにデバイス常駐サービスをエクスポートすることができると仮定する。本明細書で説明するコンテインメント・フレームワークは、携帯情報端末、ハンドヘルド・コンピュータ、高性能携帯電話などの小フットプリント・デバイスのサービスおよびアプリケーションを、ネットワーク・サービス・フェデレーションと統合するために必要なインタフェースを提供することができる。

【００９８】図７に図示し、以下により詳細に説明するように、コンテインメント・フレームワーク１４４は、所有するタイプのルックアップ・サービス１４６を有する。コンテインメント・フレームワーク１４４内のルックアップ・サービス１４６は、上述のローカル・ネットワーク・ルックアップ・サービスと類似した方法で動作することができ、発見、結合、ルックアップ、およびサービス呼び出しプロセスを利用する。たとえば、パーソナル・オーガナイザ・アプリケーション１５２は、カレンダ・サービス、コンタクト・リスト・サービス、ブックマーク・サービスなど、様々なサービスを利用することができる（図示せず）。パーソナル・オーガナイザ・アプリケーション１５２は、これらのサービスと通信するための参照を、コンテインメント・フレームワーク・ルックアップ・サービス１４６を介して得ることができる。

【００９９】コンテインメント・フレームワーク１４４は、所有するルックアップ・サービス１４６を、図７のローカル・ネットワーク・ルックアップ・サービス１３６などのオフ・デバイス・ルックアップ・サービスに統合することができる。このように、プリント・サービス１３８およびウェブ・サービス１４０などのオフ・デバイス・サービスを、コンテインメント・フレームワークのアプリケーション／サービス１４８、１５０および１５２で使用可能にすることができ、その逆に使用することもできる。たとえば、パーソナル・オーガナイザ・アプリケーション１５２は、プリント・サービスをコンテインメント・フレームワーク・ルックアップ・サービス１４６から要求することができる。コンテインメント・フレームワーク・ルックアップ・サービス１４６はまず、オン・デバイスプリント・サービスを探索することができる。これが見つからなかった場合、コンテインメント・フレームワーク・ルックアップ・サービス１４６は、プリント・サービスをネットワーク・ルックアップ・サービス１３６から要求することができる。次いで、プリント・サービス１３８のサービス・オブジェクトをパーソナル・オーガナイザ１５２へ返すことができる。

オン・デバイス・サービス／アプリケーションとオフ・デバイス・サービスの間のインタフェース１４２を、図

25

７に示す。オン・デバイス／オフ・デバイス・サービスの統合を実装する方法について、以下に詳述する。

【０１００】上述のように、サービスのクライアントはそれ自体を他のクライアントへのサービスにすることができる。たとえば、図７の高性能携帯電話の電子メール・クライアント「アプリケーション」１５０は、それ自体を、コンテインメント・フレームワーク１４４で動作するクライアントまたはネットワーク・クライアントへのサービスにすることができる。たとえば、誤動作が起こった場合、図７のプリンタ１３０が電子メール・サービスを要求して、診断情報をサービス技術者へ送信できるようにすることができる。ネットワーク・ルックアップ・サービス１３６がネットワーク・ベースの電子メール・サービスを見つけることができなかった場合、インタフェース１４２を介して電子メール・サービスを高性能携帯電話１３４から要求することができる。コンテインメント・フレームワーク１４４で動作する電子メール・アプリケーション／サービス１５０のサービス・オブジェクトを、要求側のプリンタ・クライアント１３０へ渡すことができる。この例では、プリンタ・クライアント１３０が直接電子メール・アプリケーション／サービス１５０と通信して、診断情報を含む電子メールをプリンタ・サービス技術者へ送信することができる。電子メール・アプリケーション／サービス１５０は、電子メール・サーバ・サービスを見つけることができた場合に電子メールを直ちに送信することができ、あるいは、後に携帯電話のユーザが異なるネットワークへ接続する際にこのようなサービスが使用可能になるときに、この電子メールを送信することができる。

【０１０１】上記の説明は、Ｊｉｎｉ™技術など、特定のプロトコルおよびプログラミング・モデルを参照するが、これらの特定の技術は例示的なものにすぎないことに留意されたい。たとえば、コンテインメント・フレームワーク内のアプリケーションおよびサービスを、様々なタイプの規格、プロトコル、およびプログラミング・モデルを使用するクライアント、サービス、デバイス、ネットワークなどと統合することができる。これらの規格、プロトコル、およびプログラミング・モデルには、Ｊｉｎｉ™、ＣＯＲＢＡ、ＣＯＭ／ＤＣＯＭ、Ｂｌｕｅｔｏｏｔｈ、ＣＡＬ、ＣＥＢｕｓ、ＨＡＶｉ、Ｈｏｍｅ ＡＰＩ、ＨｏｍｅＰＮＡ、ＨｏｍｅＰｎＰ、ＨｏｍｅＲＦ、ＶＥＳＡなどが含まれるが、それだけに限定されるものではない。

【０１０２】図１２－コンテインメント・フレームワークのブロック図

図１２は、コンテインメント・フレームワーク環境の基本アーキテクチャを示す抽象的なブロック図である。上述のように、コンテインメント・フレームワークはアプリケーションおよびサービス用のコンテインメント・システムを提供する。これらのアプリケーションおよびサ

26

ービスは、システム内でモジュールと呼ばれるユニットとして管理される。コンテインメント・フレームワークは軽量であり、一実施形態では、モジュールが、すべてのモジュール管理を実行する単一のフレームワーク・マネージャと対話することができる。このマネージャを、本明細書では中央フレームワーク・インスタンスと呼ぶ。一実施形態では、中央フレームワーク・インスタンスをＪａｖａ™クラスのインスタンスとして実装することができる。図１２は、中央フレームワーク・インスタンス１７０、およびそれが含む／管理するコードおよびデータである。図１２がコンテインメント・フレームワークの一実施形態を示すことに留意されたい。他の実施形態は、異なるアーキテクチャを使用することができ、かつ／または異なるプログラミング言語またはソフトウェア環境で実装することができる。たとえば、図１２の中央フレームワーク・インスタンス１７０が実行するモジュール管理／コンテインメントを、他の実施形態では、多数のオブジェクトまたはコンポーネントが実行することができる。

【０１０３】図１２のように、中央フレームワーク・インスタンス１７０は、現在システムにロードされているモジュールを表現するデータ１８２を含む。コンテインメント・フレームワーク・アーキテクチャは非階層である。したがって、ロードされたモジュールを、モジュールの単層リストまたは配列として表現することができる。この非階層システムは、コア・コンテインメント・フレームワーク・コードおよびフレームワーク内で動作するモジュールを小さく保つための助けとなる。ＪａｖａＢｅａｎｓ™コンポーネントなど、階層コンポーネントを使用するシステムは、それに伴う利点を提供することができるが、これらの利点は、より多くのシステム・リソースを必要とするより複雑な管理システムを負担することで得られる。しかし、コンテインメント・フレームワークは、非階層モジュールが階層コンテインメント・システムの利点を多数得るための機構を提供する。この機構を、図１３および図１４について以下に説明する。

【０１０４】図１２のように、一実施形態では、中央フレームワーク・インスタンス１７０が、公開されたアクセス可能なメソッド１７８を含み、モジュールがこれらを呼び出すことができる。これらのメソッドを抽象グループに分割することができる。たとえば、あるグループのメソッド１７２には、ルックアップ・メソッドを含めることができる。ルックアップ・メソッドは、上述のルックアップ・サービス機能性を実装する。モジュールは、モジュール記述子を中央フレームワーク・インスタンス１７０のルックアップ・メソッドに渡して、特定のサービス・モジュールを探し出すことができる。コンテインメント・フレームワーク・ルックアップ・プロセスについては、以下で図１６について説明する。別のグル

ープのフレームワーク・メソッド１７４には、モジュールをロードおよびアンロードするメソッドを含めることができる。サービス・モジュールを見つけた後、クライアント・モジュールは中央フレームワーク・インスタンス１７０に対して、このサービス・モジュールをロードして、ロードしたモジュールへの参照を返すように要求することができる。次いで、クライアント・モジュールがこのサービスを呼び出すことができる。クライアントは、サービス・モジュールを使用し終えたときに、フレームワーク・メソッドを呼び出してこのサービス・モジュールを解放することができる。異なるグループとして記載したが、メソッドをルックアップおよびロード／アンロードのグループに分割することは、概念上の分割にすぎないことがある。たとえば、一実施形態では、ルックアップ・メソッドが、合致するモジュールをロードして、合致したモジュールへの参照を返すこともできる。

【０１０５】図１２は、フレームワーク・メタデータと呼ばれるシステム・データ１８０も示す。これは、ロードしたモジュールのリストを記述するデータ１８２およびシステムの状態を記述する他のデータを含むことができる。中央フレームワーク・インスタンス１７０の別の抽象グループのメソッド１７６には、リフレクション・メソッドを含めることができる。リフレクション・メソッドは、コア・メタデータ１８０への直接アクセスを提供するので、他のグループのメソッドとはやや異なる。システム・モジュールと呼ばれるモジュールの特殊クラスが、リフレクション・メソッドを呼び出して、メタデータ１８０へのアクセスを得ることができる。通常のモジュールはメタデータ１８０にアクセスすることができない。

【０１０６】コア・システム・データ１８０への参照を受け取った後、システム・モジュールはいかなる望ましい方法でもデータを使用あるいは修正することができる。したがって、コンテインメント・フレームワークは大幅に拡張可能である。中央フレームワーク・インスタンス１７０はそれ自体を小さく保つことができ、システム・モジュールを追加して、中央フレームワーク・インスタンス１７０によってまだ使用可能にされていない機能性を実装することができる。たとえば、システム・モジュールは、図７から図１１について上述した、コンテインメント・フレームワーク内で動作するアプリケーション／サービスと外部ネットワークに基づくサービスの間の統合を、可能にすることができる。

【０１０７】この例では、このようなシステム・モジュールを、外部ネットワークのプロトコルおよびプログラミング・モデルに従う第２のルックアップ・サービスとして書くことができる。たとえば、Ｊｉｎｉ™ネットワークでは、Ｊｉｎｉ™ネットワーク・ルックアップ・サービスを発見してネットワーク・ルックアップ・サービスを結合し、それ自体を第２のルックアップ・サービスとして登録するシステム・モジュールを書くことができる。ネットワーク・クライアントがサービスを要求したとき、ネットワーク・ルックアップ・サービスが、システム・モジュールによって実装されたルックアップ・サービスを呼び出すことができる。このシステム・モジュールは、コンテインメント・フレームワーク内で、要求したサービスの記述に合致するサービス・モジュールを発見しようと試みることができる。合致が見つかった場合、システム・モジュールは、このサービス・モジュールをネットワーク・クライアントへエクスポートするために必要なステップを実行することができる。これは、システム・モジュールがシステム・モジュール・リストおよびメタデータへの完全アクセスを有しているからである。たとえば、システム・モジュールは、合致したサービス・モジュールをシステムへロードして登録し、Ｊａｖａ™インタフェースなど、新たにロードしたモジュールへのインタフェースを要求側に返すことができる。

【０１０８】図１３および図１４－階層環境のシミュレート

モジュールの階層コンテキストを確立することは、しばしば望ましい。たとえば、同じタイプのいくつかのサービス・モジュールがシステム内に存在することができるが、それぞれがわずかに異なる挙動を示すことができる。階層コンテインメント・システムでは、モジュールによるサービスの要求を、要求側モジュールの親または含んでいるモジュールを通じてフィルタリングすることができ、特定のサービス・モジュールへの参照を要求側へ戻すことができるようにする。階層コンテインメントは、他の固有の利点も有する。たとえば、データをモジュールの階層の間で容易に分散させ、格納する能力などである。しかし、上述のように、階層コンテインメント・システムの完全実装には、メモリおよび処理能力など、必要なシステム・リソースに関して非常に費用がかかることがある。コンテインメント・フレームワークは、開発者とアプリケーションに、階層コンテインメントの利点を多数もたらす機構を提供することができるが、通常、これに伴う高い間接経費はかからない。

【０１０９】たとえば、一実施形態のコンテインメント・フレームワークでは、モジュールがそれ自体を、他のモジュールのモジュール要求リスナとして登録することができる。たとえば、モジュールＡは、それ自体を、モジュールＢの要求リスナとして登録することができる。これは、たとえば、中央フレームワーク・インスタンスのＡｄｄＲｅｑｕｅｓｔＬｉｓｔｅｎｅｒメソッドを呼び出すことによって行う。続いて、モジュールＢが、特定のサービスを見つけるために中央フレームワーク・インスタンスのメソッドを呼び出したとき、中央フレームワーク・インスタンスは、モジュールＢのモジュール要求リスナがあるかどうかを検査する。この場合、中央フレームワーク・インスタンスはモジュールＡを要求リス

ナとして見つけ、モジュールＡに対して、要求されたサービス・モジュールをモジュールＢへ提供するように求める。

【０１１０】図１３および図１４は、コンテインメント・フレームワークにおけるモジュール要求リスナの例示的使用である。図１３は、プリント・サービスの所望の概念上のモジュール階層である。図示のように、２つのプリント・サービス・モジュール１９２および１９４、プリント・サービスＡおよびプリント・サービスＢが、プリント・マネージャ・モジュール１９０内にカプセル化されている。たとえば、２つのプリント・サービス１９２および１９４は、異なる場所に印刷することができ、異なる解像度および色機能などを有することができる。これらのプリント・サービス・モジュールのいずれかが、別のモジュールによるプリント・サービスのルックアップ要求を満たすことができる。しかし、特定のプリント・サービスを選択して返すプリント・マネージャ・モジュールを使用することが望ましいことがある。たとえば、プリント・マネージャ１９０は、どのクライアント・モジュールが印刷要求を行うかに基づいてプリント・サービスを選択することができ、あるいは、プリント・マネージャは、ユーザに対して所望のプリント・サービス特性を入力するように求めるダイアログ・ボックスを表示することができる。

【０１１１】コンテインメント・フレームワークは非階層コンテインメント・モデルを利用するが、図１３の階層を実現することができる。これは、プリント・マネージャ・モジュール１９０を、プリント・サービスを要求することができるクライアント・モジュールのモジュール要求リスナとして登録することによって行う。図１４は、システム内で動作することができるモジュール例１９８である。先に述べたように、これらのモジュールはみずから他のモジュールをサービスとして使用することができる。コンテインメント・フレームワークの非階層モデルに従って、モジュールを、単層のレイアウトで固有のモジュール階層を使わずに配列して示す。

【０１１２】この例では、ウェブ・ブラウザ・モジュール１９６が、たとえばウェブ・ページを印刷するために、印刷要求を行うように動作可能にすることができる。図１４のように、プリント・マネージャ・モジュール１９０を、ウェブ・ブラウザ・モジュール１９６のモジュール要求リスナとして登録することができる。プリント・サービス要求をウェブ・ブラウザ１９６から受け取ると、コンテインメント・フレームワーク・ルックアップ・サービスは、ウェブ・ブラウザ・モジュール１９６の要求リスナとして登録されたプリント・マネージャ・モジュール１９０を見つけることができ、プリント・マネージャ・モジュール１９０に対して、プリント・サービス・モジュールをウェブ・ブラウザ要求側１９６へ提供するように求めることができる。次いで、プリント・マネージャ・モジュール１９０が、プリント・サービス・モジュールＡ１９２またはプリント・サービス・モジュールＢ１９４への参照を返すことができ、あるいは、プリント・マネージャ・モジュール１９０が、ダイアログ・ボックスをユーザに提示して、どのプリント・サービス・モジュールを返すかを判断することができる。したがって、図１３の所望のモジュール階層を、コンテインメント・フレームワークの非階層モジュールのために実装することができる。

【０１１３】図１５－パーセル・パッケージ化ユニットモジュールを、パーセルと呼ばれるユニットにパッケージ化することができる。このパッケージ化はいくつかの目的に役立つ。たとえば、パーセルは、関連するコードおよびデータをユニットとして管理するために好都合な機構を提供する。密接に関連したモジュールが静的な従属性を有する場合、これらを共にパーセルにパッケージ化することができる。パーセルを使用して、システム内のインストレーションおよびアップグレードを処理することができる。

【０１１４】図１５は、個人情報マネージャ（ＰＩＭ）に関するモジュールを共にグループ化するパーセル例２００である。この図では、カレンダ・モジュール２０２、コンタクト・リスト・モジュール２０４、アポイントメント・モジュール２０８、およびユーザ・インタフェース・モジュール２０６を示す。他の様々なモジュールを、所望するようにパーセル内に存在させることができる。ＰＩＭパーセル２００のモジュールは、ブックマーク・サービス、検索サービスなど、コンテインメント・フレームワーク内で動作する様々なコア・サービス・モジュールを使用することもできる。ＰＩＭパーセルを使用すると、ＰＩＭアプリケーションのインストレーションおよびアップグレードを簡素化することができる。ＰＩＭモジュールをこのようにしてパーセルにパッケージ化すると、多目的な開発のために別々のコード・ユニットを作成する際の開発時間の利点もある。

【０１１５】パーセルは、非階層モジュールのランタイム・コンテキストを提供するための追加の方法も提供する。モジュールがシステムにロードされたとき、中央フレームワーク・インスタンスは、このモジュールが属するパーセルがあればそれを指定するメタデータを格納することができる。サービス・モジュールは後にこの情報を使用して、異なるクライアント・モジュールに対し、そのクライアントが属するパーセルに応じて異なる方法でサービスを提供することができる。たとえば、クライアント・モジュールは、ファイル・アクセス・サービス・モジュールを使用してルート・ディレクトリを得ることができる。ファイル・アクセス・モジュールは、異なるクライアントに対し、そのクライアントが属するパーセルに応じて異なるルート・ディレクトリを返すことができる。

【０１１６】図１６－モジュール要求フローチャート
図１６は、典型的なルックアップ・プロセスを示すフローチャートであり、このプロセスでは、中央フレームワーク・インスタンスが、サービス・モジュールへのルックアップ要求をクライアント・モジュールから受け取ったときに実行することができる。図１６が例示的なものであり、様々なステップを結合し、省略し、あるいは修正することができることに留意されたい。たとえば、先に述べたように、ルックアップ・プロセスをカスタマイズするシステム・モジュールを追加することができる。

【０１１７】図１６のステップ３００で、中央フレームワーク・インスタンスがモジュール・ルックアップ要求を要求側モジュールから受け取る。たとえば、要求側モジュールは、中央フレームワーク・インスタンスのＲｅｑｕｅｓｔＭｏｄｕｌｅメソッドを呼び出して、要求されているサービス・モジュールのモジュール記述子、ならびに要求側モジュール自体への参照を渡すことができる。要求側モジュールへの参照をシステム・データに追加して、サービス・モジュール・ユーザの追跡を維持することができる。以下でより詳細に述べるように、他のモジュールがモジュールを使用していないときは、このモジュールをアンロードすることができる。

【０１１８】要求側モジュールによって渡されたモジュール記述子は、要求したモジュールについての様々な属性を指定し、フレームワーク・インスタンスがこの属性を使用して、合致するモジュールを見つけようと試みることができる。このモジュール記述子は、要求したモジュールのサービス・タイプ、クラス名、および／またはサービス固有の属性などの情報を含むオブジェクトにすることができる。要求側はテキスト記述を中央フレームワーク・インスタンスに渡すこともでき、中央フレームワーク・インスタンスはこの記述を使用して、モジュール記述子オブジェクトを作成することができる。

【０１１９】ステップ３０２で、中央フレームワーク・インスタンスは、要求リスナ・モジュールが要求モジュールについて登録されているかどうかを検査する。要求リスナが見つかった場合、ステップ３０４で、フレームワーク・インスタンスが要求リスナにこの要求を通知し、要求リスナに対して、モジュール要求記述子に合致するモジュールを提供しようと試みるように指令する。要求リスナが合致するモジュールを提供できた場合、実行はステップ３１４へ進行する。そうでない場合は、登録されている他の要求リスナに対して、合致が見つかるかあるいはそれ以上の要求リスナがなくなるまで、モジュールを提供するように求めることができる。

【０１２０】要求リスナが見つからなかった場合、あるいはどの要求リスナも要求されたモジュールを提供できなかった場合、実行はステップ３０６へ進行する。しかし、一実施形態では、１つまたは複数の要求リスナが要求モジュールについて登録されており、そのうちのいずれも合致するモジュールを提供できなかった場合、実行はステップ３０４の後で停止することがある。ステップ３０６で、中央フレームワーク・インスタンスがモジュールのリストを検査して、モジュールの１つがモジュール記述子に合致するかどうかを判定する。合致が見つかった場合、ステップ３０８で、フレームワーク・インスタンスが、合致したモジュールがマルチ・インスタンス化可能かどうかを検査する。そうでない場合、実行はステップ３１４へ進行する。

【０１２１】ステップ３０８で、合致したモジュールがマルチ・インスタンス化可能であることが分かった場合、中央フレームワーク・インスタンスは合致を求めてモジュール・リストの探索を継続することができる。探索するモジュールがそれ以上なくなった場合、実行はステップ３１０へ進行する。ステップ３１０で、フレームワーク・インスタンスがモジュール提供側モジュールをモジュール・リスト内で探索する。モジュール提供側モジュールは、要求されたモジュールを提供することができるモジュールである。たとえば、ネットワーク・ルックアップ・サービスを、コンテインメント・フレームワーク用のモジュール提供側モジュールとしてインポートすることができる。

【０１２２】モジュール提供側モジュールが見つかった場合、ステップ３１２で、中央フレームワーク・インスタンスがモジュール提供側モジュールに要求を通知し、モジュール要求記述子に合致するモジュールを提供しようと試みるように指令する。合致が見つかった場合、実行はステップ３１４へ進行する。モジュール提供側が要求されたモジュールを提供できなかった場合、中央フレームワーク・インスタンスは他のモジュール提供側モジュールを探索して、ステップ３１２を繰り返すことができる。モジュール提供側がモジュール・リスト内に存在しない場合、あるいはいずれも要求されたモジュールを提供できなかった場合、要求側に、この要求を満たすことができないことが通知され、実行が完了する。

【０１２３】ステップ３１４には、ステップ３０４、３０８または３１２から到達することができる。すべての場合において、モジュール要求記述子に合致するモジュールが見つけられる。ステップ３１４で、要求側は合致したモジュールのユーザとして登録され、ステップ３１６で、合致したモジュールへの参照が要求側に返される。合致したモジュールのロードおよび初期化に含まれる、いかなる必要な初期化ステップも、ステップ３１４で実行される。たとえば、モジュールは、モジュールがロードされたときに呼び出されるＩｎｉｔｉａｌｉｚｅメソッドを有することができる。

【０１２４】上述のように、図１６のフローチャートは例示的なものであり、様々な実施形態が異なるルックアップ／ロード・シナリオを有することができる。たとえば、モジュールは、中央フレームワーク・メソッドを呼

び出して、合致したモジュールへの参照を返すことなくサービス・モジュールをロードすることができ、あるいはいくつかの場合で要求リスナを無視することができる。

【０１２５】図１７－モジュール解放フローチャート
クライアント・モジュールがサービス・モジュールを使用し終えたとき、クライアントは中央フレームワーク・インスタンスのメソッドを呼び出して、このモジュールを解放することができる。図１７は、モジュール解放プロセスを示すフローチャートである。図１７のフローチャートは例示的なものであり、異なる実施形態について、必要であるように、あるいは所望するように、様々なステップを結合し、省略し、追加し、あるいは修正することができる。

【０１２６】ステップ３３０で、中央フレームワーク・インスタンスは、モジュール解放通知をユーザ・モジュールから受け取る。図１６について先に述べたように、ユーザ・モジュールがサービス・モジュールを要求したとき、ユーザ・モジュールがサービス・モジュールのユーザのリストに追加される。ステップ３３２で、中央フレームワーク・インスタンスが、解放するユーザ・モジュールを、解放されるモジュールのユーザのリストから削除する。ステップ３３４で、フレームワーク・インスタンスは、他のユーザ・モジュールがこの解放されるモジュールを使用中であるかどうかを判定する。この判定は、たとえば、他のモジュールが解放モジュールのユーザ・モジュール・リスト内に存在するかどうかを検査することによって行う。そうである場合、実行が停止する。

【０１２７】他のモジュールが、解放されるモジュールを使用中でない場合、中央フレームワーク・インスタンスは、解放されるモジュールをアンロードしようと試みることができる。ステップ３３６で、フレームワーク・インスタンスは、解放されるモジュールのＣａｎＦｉｎａｌｉｚｅメソッドを呼び出すことができる。ＣａｎＦｉｎａｌｉｚｅメソッドは、このモジュールをアンロードできる場合にｔｒｕｅを返し、そうでない場合にｆａｌｓｅを返す。ステップ３３６で、ＣａｎＦｉｎａｌｉｚｅメソッドがｆａｌｓｅを返した場合、実行が停止する。そうでない場合、解放されるモジュールのＦｉｎａｌｉｚｅメソッドを呼び出すことができる。Ｆｉｎａｌｉｚｅメソッドは、リソースの解放など、モジュールのアンロードに必要ないかなるステップも実行することができる。次いで、このモジュールをアンロードすることができ、特定の実施形態に応じてこれにガベージ・コレクションなどを含めることができる。

【０１２８】本発明を特定の実施形態と共に説明したが、本明細書に述べた特定の形式に限定することを意図するものでなく、逆に、添付の特許請求の範囲によって定義された本発明の精神および範囲内に適度に含めることができるような代替物、修正および均等物を包含することを意図するものである。

【図面の簡単な説明】

【図１】ブックマークを付けることができる様々なタイプのデータ・ソースを示すブロック図である。

【図２】特定のデータ・ソースを参照するブックマークを作成するプロセスを示すフローチャートである。

【図３】それぞれのデータ・ソースを参照するブックマークを使用するプロセスを示すフローチャートである。

【図４】ＪａｖａＢｅａｎｓ™起動フレームワークアーキテクチャを含む主要な要素を示すブロック図である。

【図５】典型的な小フットプリント・デバイスのハードウェア・アーキテクチャを示すブロック図である。

【図６】コンテインメント・フレームワーク内のアプリケーションおよびサービスを実行するシステムに含まれているハードウェア／ソフトウェア層の典型的な階層を示す図である。

【図７】コンテインメント・フレームワーク内のアプリケーション／サービスを実行する小フットプリント・デバイスがローカルのサービス・ベースのネットワークに接続されている例示的ネットワークを示す図である。

【図８】サービス・プロバイダがルックアップ・サービスを見つける発見プロセスを示す図である。

【図９】サービス・プロバイダがそのサービスをルックアップ・サービスに登録する結合プロセスを示す図である。

【図１０】クライアントがルックアップ・サービスからサービスを要求するルックアップ・プロセスを示す図である。

【図１１】クライアントがルックアップ・サービスから受け取ったサービス・オブジェクトを使用してサービスを呼び出すサービス呼び出しプロセスを示す図である。

【図１２】コンテインメント・フレームワークの基本アーキテクチャを示す抽象的なブロック図である。

【図１３】コンテインメント・フレームワーク内のモジュール要求リスナを使用して階層コンテインメント環境をシミュレートすることを示す図である。

【図１４】コンテインメント・フレームワーク内のモジュール要求リスナを使用して階層コンテインメント環境をシミュレートすることを示す図である。

【図１５】パーセルを使用してモジュールを共にグループ化することを示す図である。

【図１６】中央フレームワーク・インスタンスがクライアント・モジュールからサービス・モジュールのルックアップ要求を受け取ったときに実行することができる典型的なルックアップ・プロセスを示すフローチャートである。

【図１７】モジュール解放プロセスを示すフローチャートである。

【符号の説明】

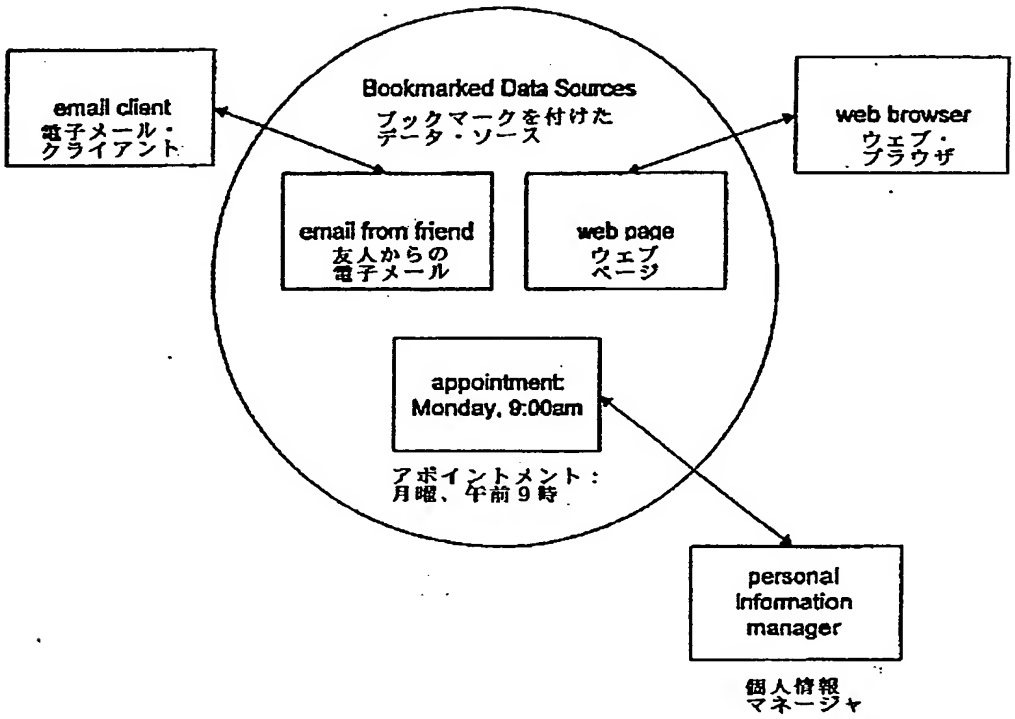
35

１３０　プリンタ
１３２　インターネット・テレビ
１３４　「高性能」携帯電話
１３６　ルックアップ・サービス
１３８　プリント・サービス
１４０　Ｗｅｂサービス
１４２　コンテインメント・フレームワーク・インタフェース

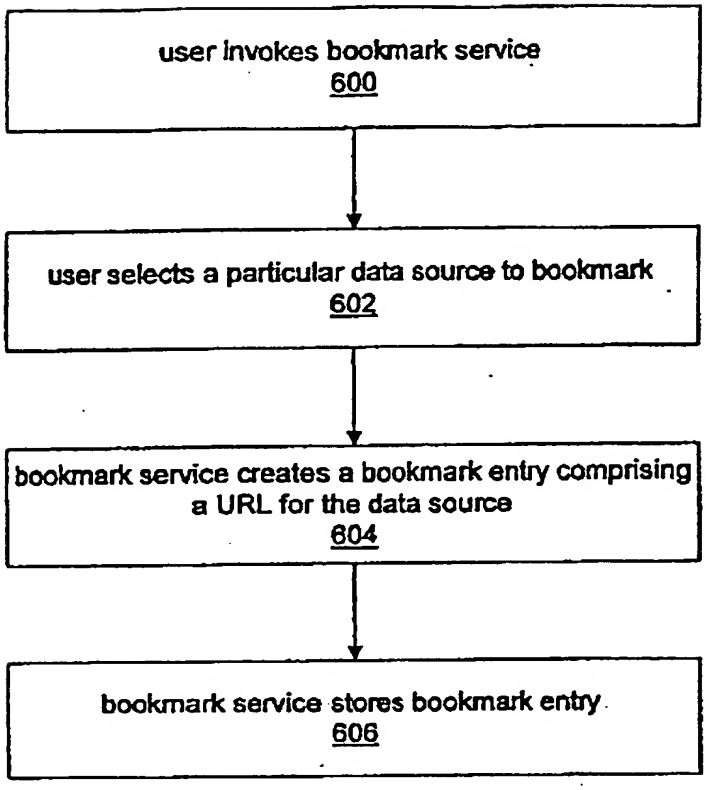
36

１４４　コンテインメント・フレームワーク
１４６　ルックアップ・サービス
１４８　Ｗｅｂブラウザ
１５０　電子メール・クライアント
１５２　パーソナル・オーガナイザ
１５４　インターネット

【図１】



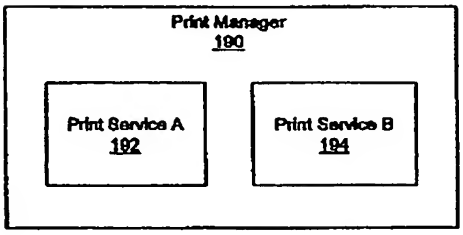
【図２】



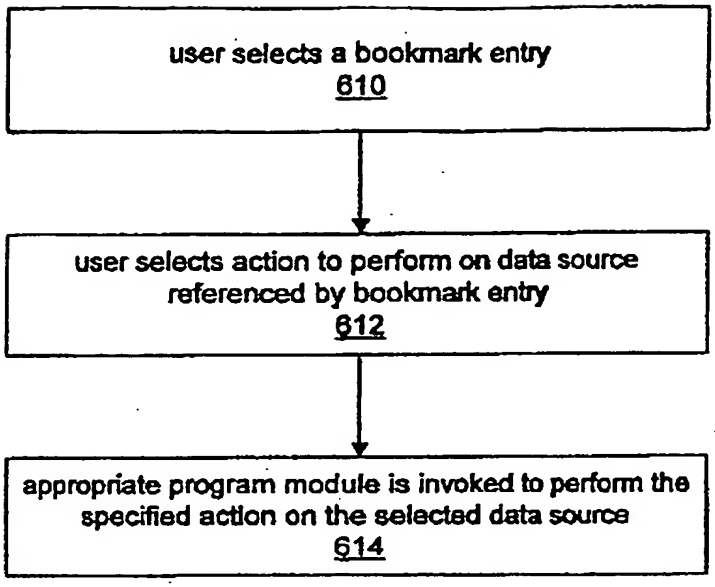
６００　ユーザがブックマーク・サービスを呼び出す
６０２　ユーザがブックマークを付ける特定のデータソースを選択する
６０４　ブックマーク・サービスが、データ・ソースのＵＲＬを含むブックマーク・エントリを作成する
６０６　ブックマーク・サービスがブックマーク・エントリを格納する

【図１３】

１９０　プリント・マネージャ
１９２　プリント・サービスＡ
１９４　プリント・サービスＢ

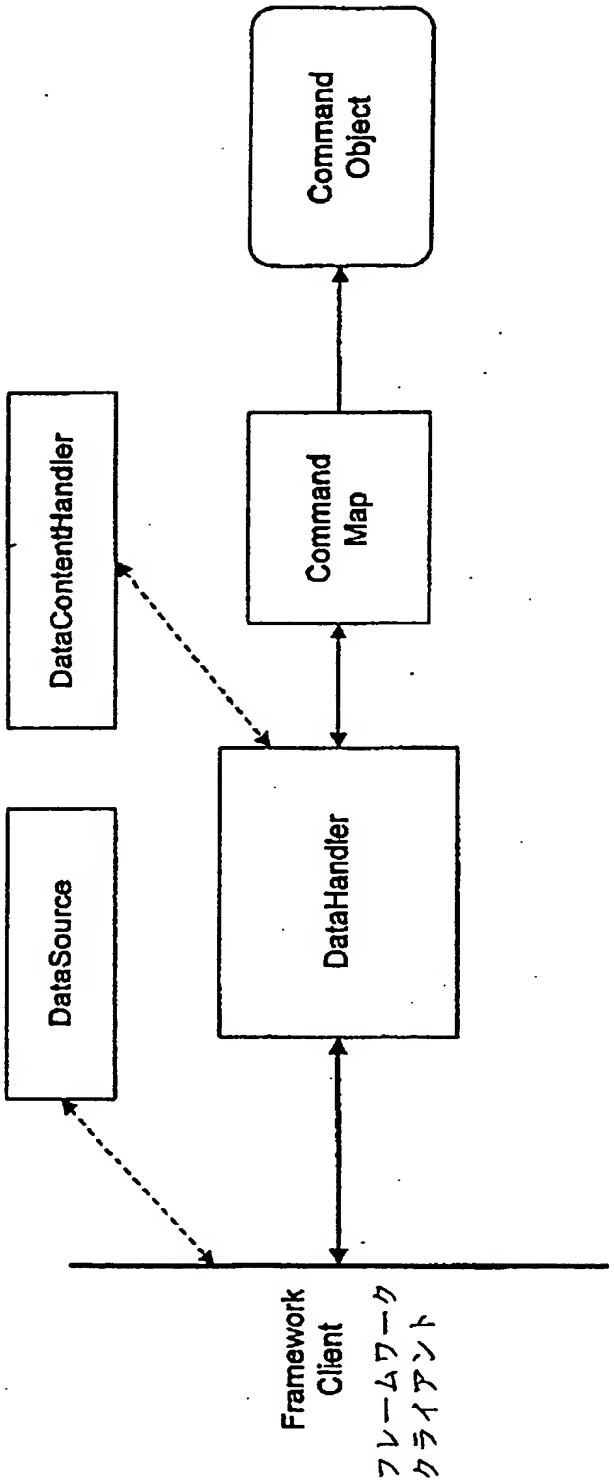


【図３】

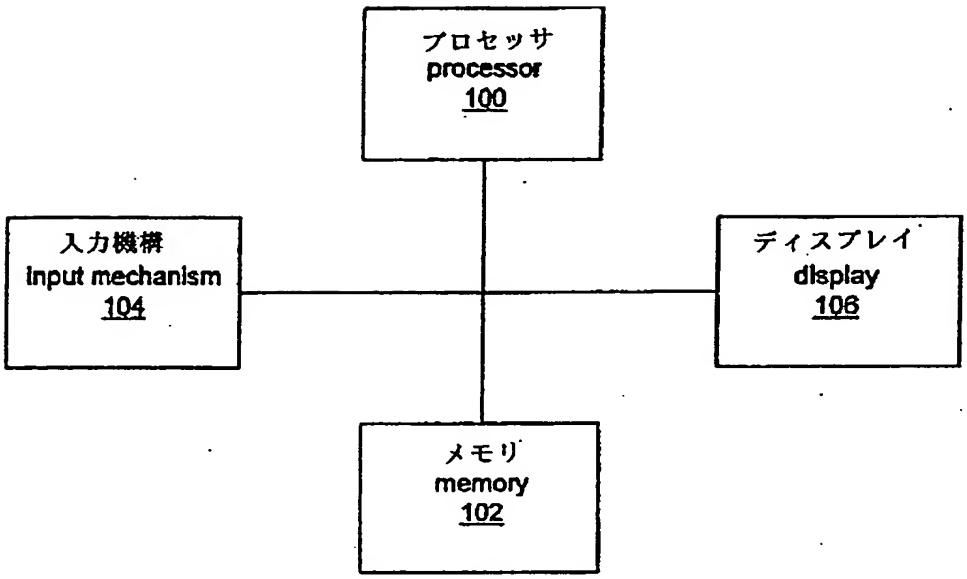


６１０　ユーザがブックマーク・エントリを選択する
６１２　ユーザが、ブックマーク・エントリによって参照されたデータ・ソースに対して実行するアクションを選択する
６１４　指定したアクションを、選択したデータ・ソースに対して実行するために、適切なプログラム・モジュールを呼び出す

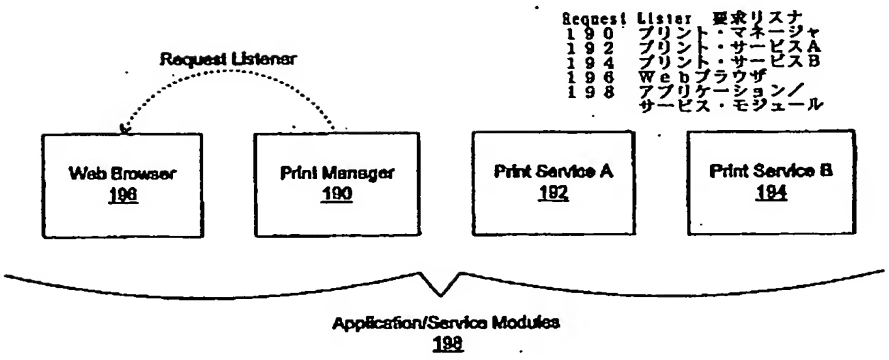
【図４】


Command Object
Command Map
DataContentHandler
DataHandler
DataSource
Framework Client
フレームワーク クライアント

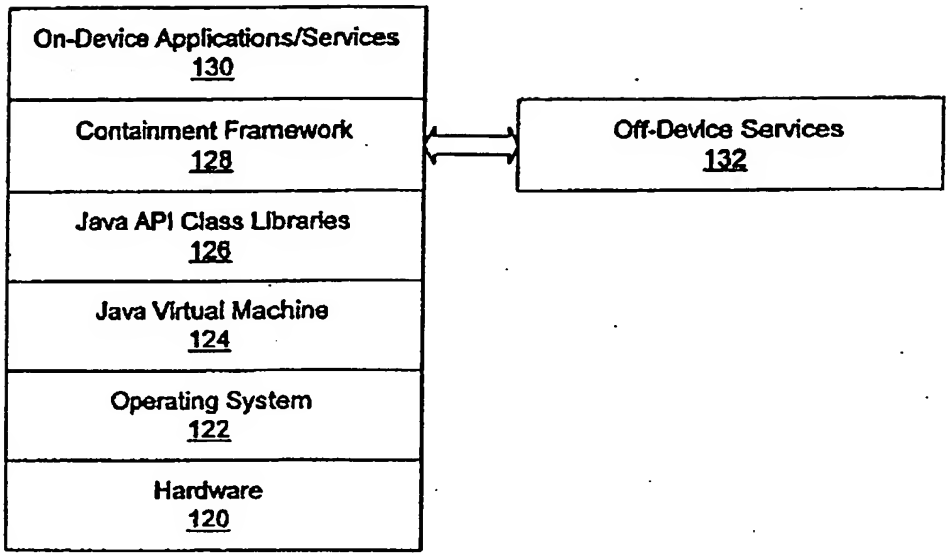
【図５】


プロセッサ
processor
100
入力機構
input mechanism
104
ディスプレイ
display
106
メモリ
memory
102

【図１４】

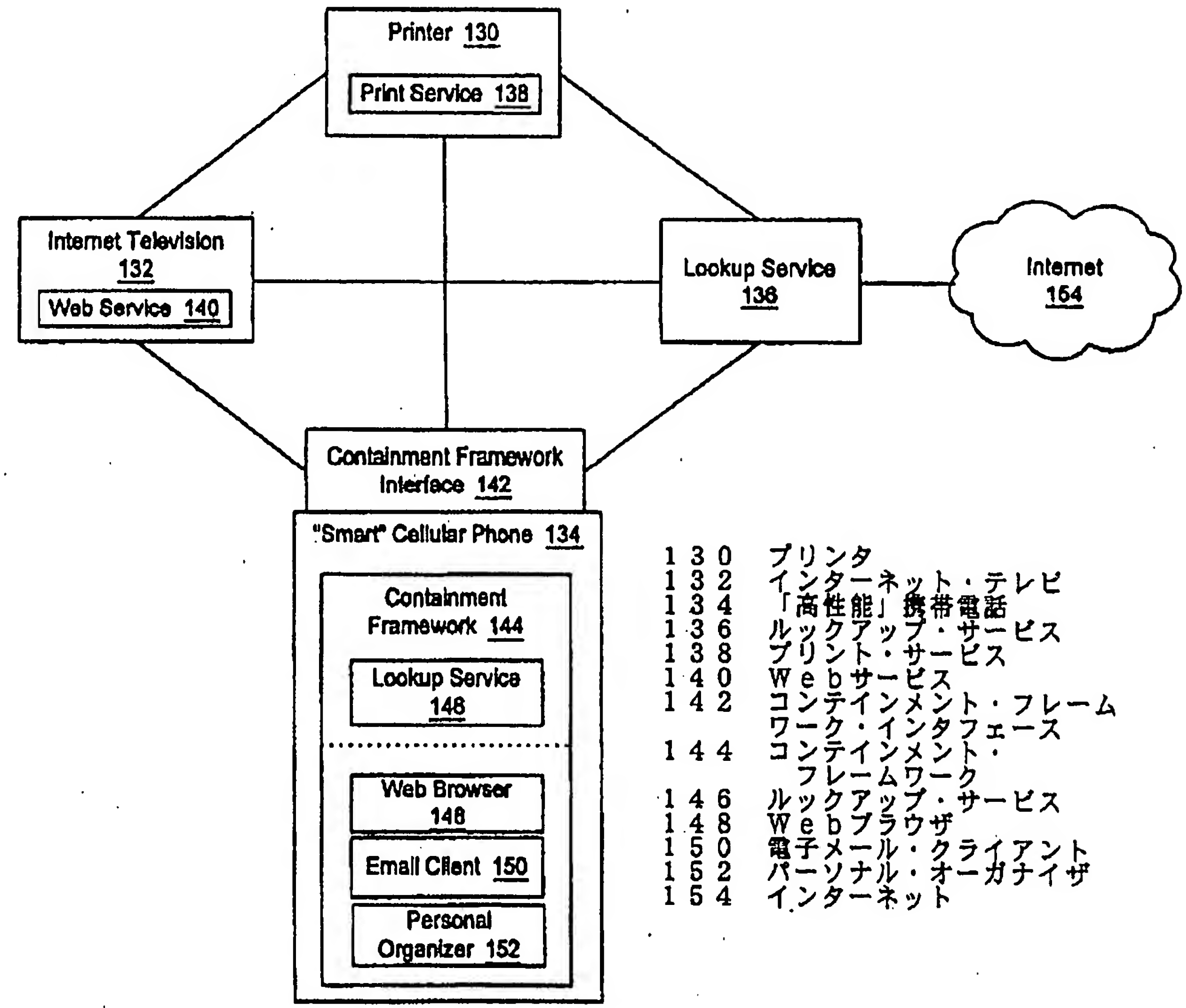

Request Lister 要求リスナ
190 プリント・マネージャ
192 プリント・サービスA
194 プリント・サービスB
196 Web ブラウザ
198 アプリケーション／
サービス・モジュール
Request Listener
Web Browser
196
Print Manager
190
Print Service A
192
Print Service B
194
Application/Service Modules
198

【図６】



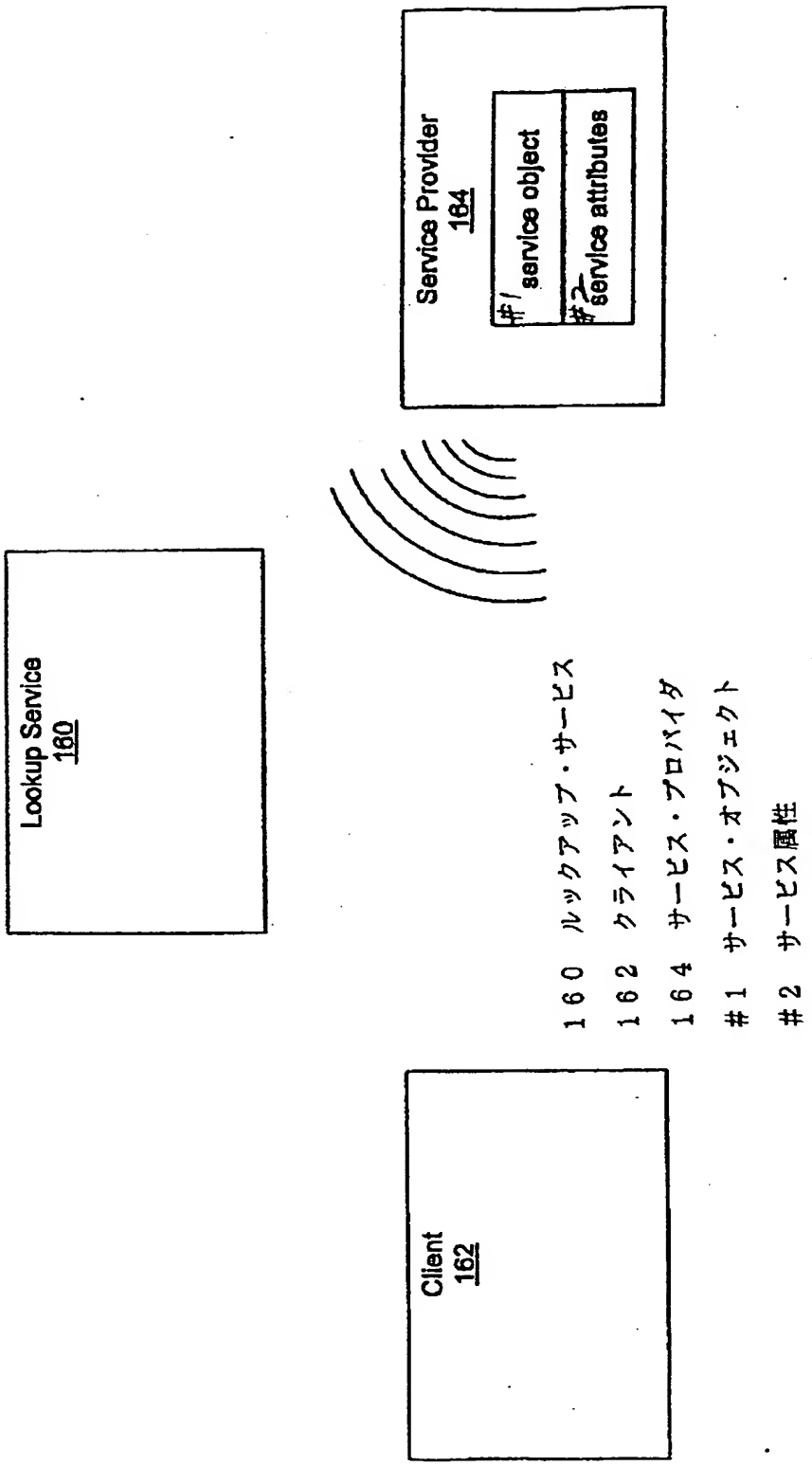
１２０　ハードウェア
１２２　オペレーティング・システム
１２４　Ｊａｖａ仮想マシン
１２６　Ｊａｖａ　ＡＰＩクラス・ライブラリ
１２８　コンテインメント・フレームワーク
１３０　オン・デバイス・アプリケーション／サービス
１３２　オフ・デバイス・サービス

【図７】

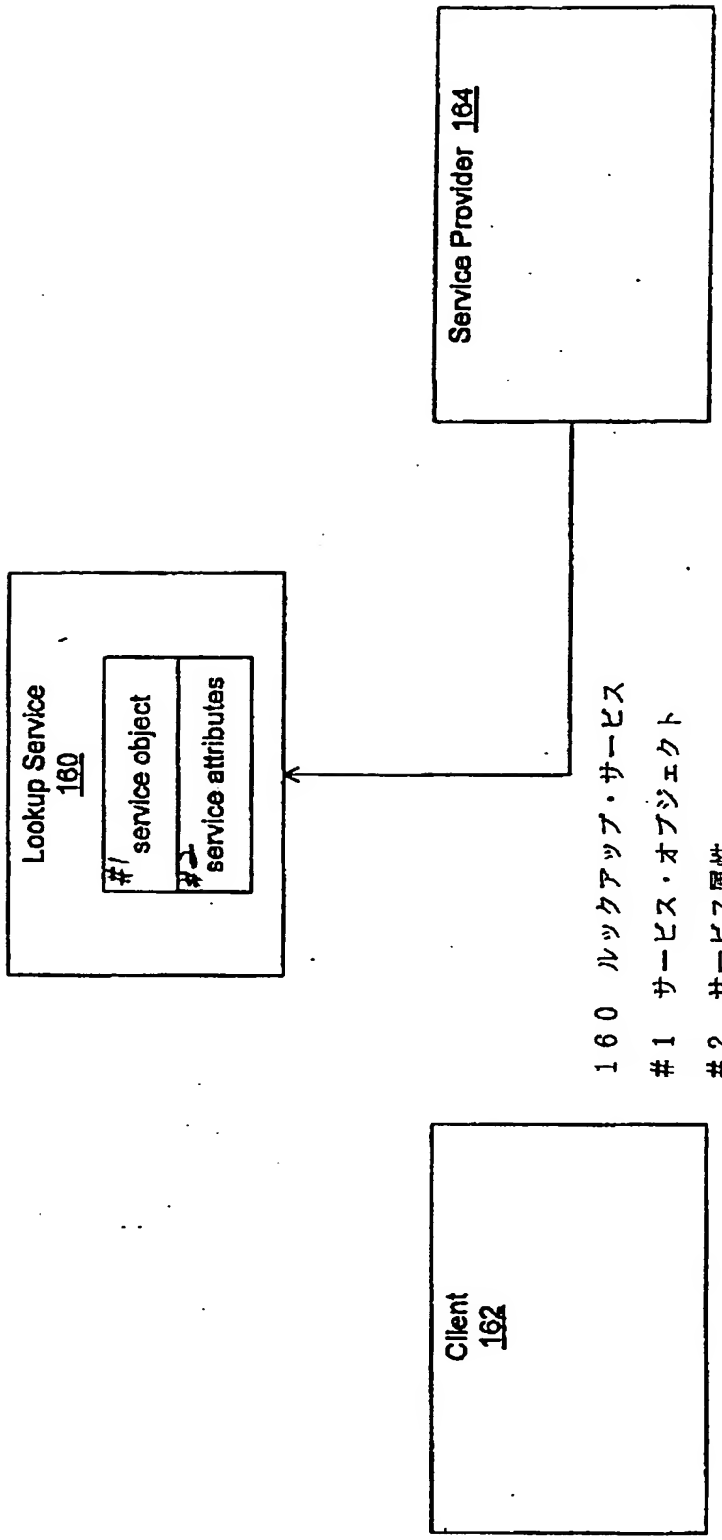


１３０　プリンタ
１３２　インターネット・テレビ
１３４　「高性能」携帯電話
１３６　ルックアップ・サービス
１３８　プリント・サービス
１４０　Ｗｅｂサービス
１４２　コンテインメント・フレームワーク・インタフェース
１４４　コンテインメント・フレームワーク
１４６　ルックアップ・サービス
１４８　Ｗｅｂブラウザ
１５０　電子メール・クライアント
１５２　パーソナル・オーガナイザ
１５４　インターネット

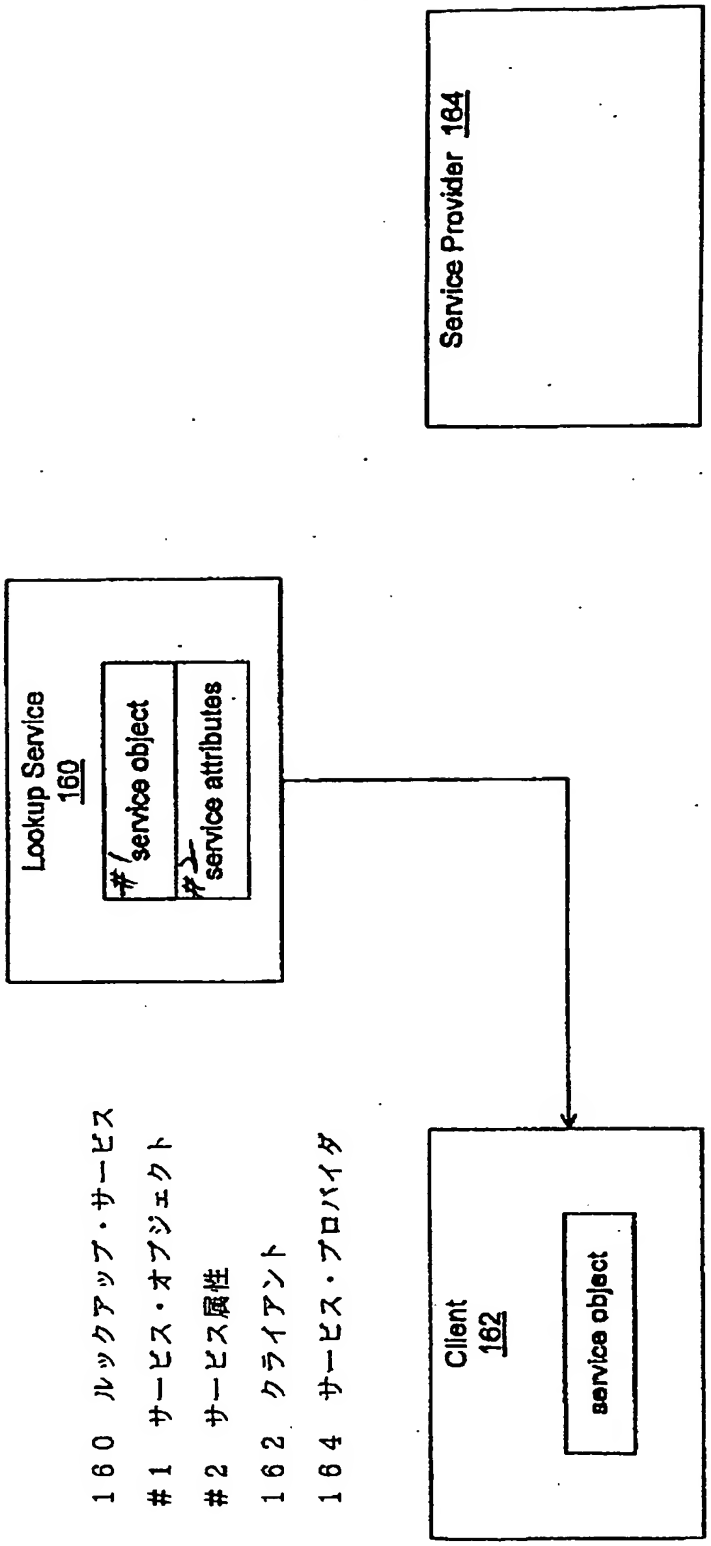
【図８】


Service Provider
184
#1 service object
#2 service attributes
Lookup Service
160
Client
162
１６０　ルックアップ・サービス
１６２　クライアント
１６４　サービス・プロバイダ
＃１　サービス・オブジェクト
＃２　サービス属性

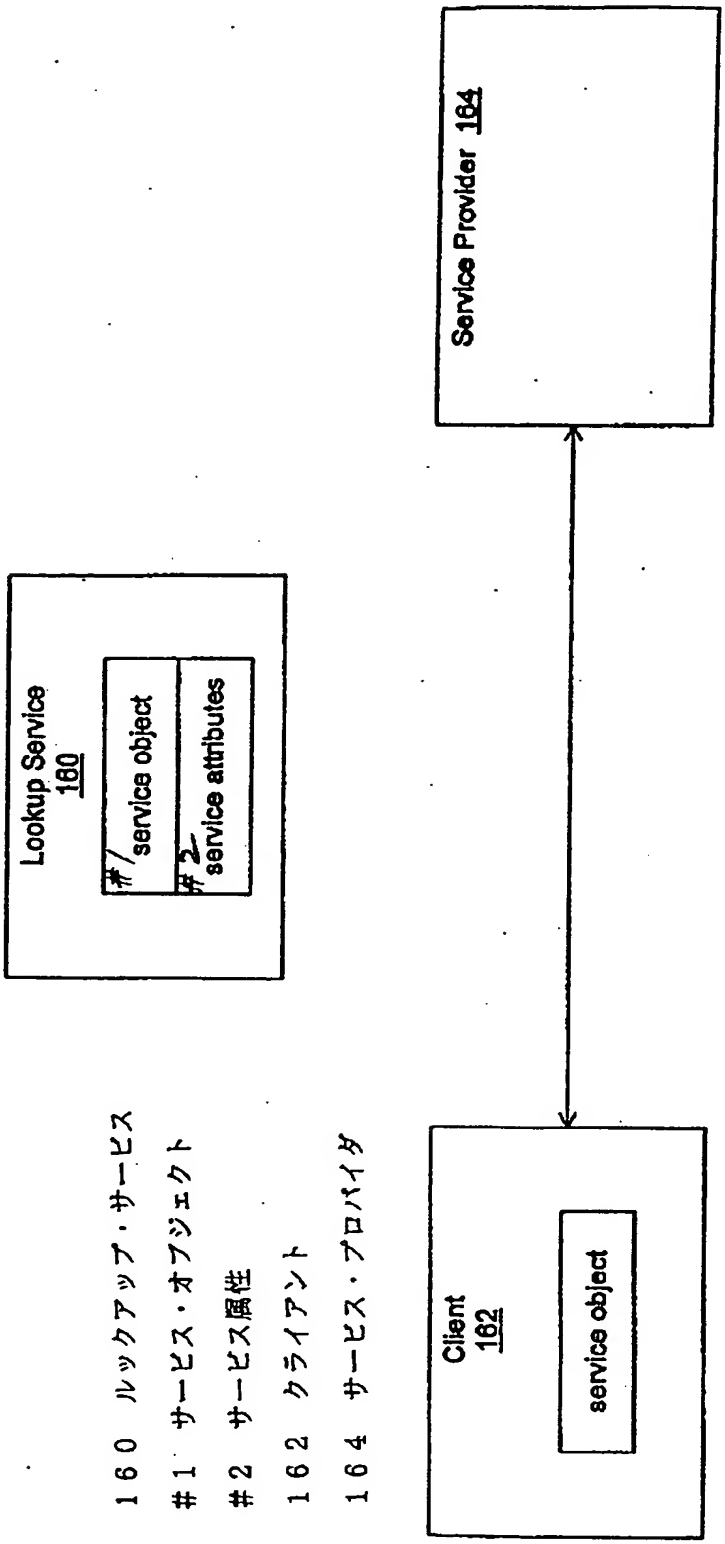
【図９】


Service Provider 164
Lookup Service 160
#1 service object
#2 service attributes
Client 162
１６０　ルックアップ・サービス
＃１　サービス・オブジェクト
＃２　サービス属性
１６２　クライアント
１６４　サービス・プロバイダ

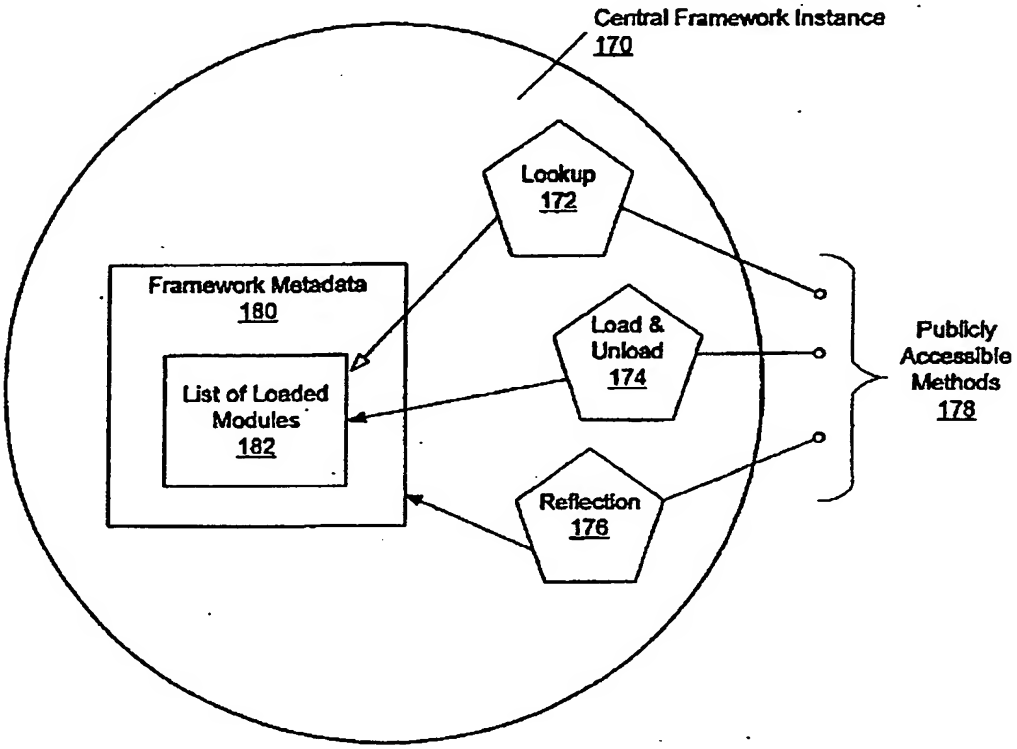
【図１０】


Service Provider 164
Lookup Service 160
#1 service object
#2 service attributes
Client 162
service object
１６０　ルックアップ・サービス
＃１　サービス・オブジェクト
＃２　サービス属性
１６２　クライアント
１６４　サービス・プロバイダ

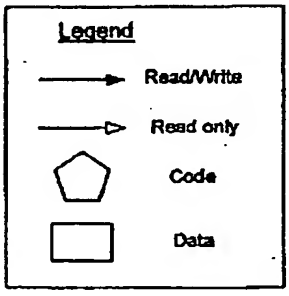
【図１１】


Service Provider 164
Lookup Service 160
#1 service object
#2 service attributes
Client 162
service object
１６０　ルックアップ・サービス
＃１　サービス・オブジェクト
＃２　サービス属性
１６２　クライアント
１６４　サービス・プロバイダ

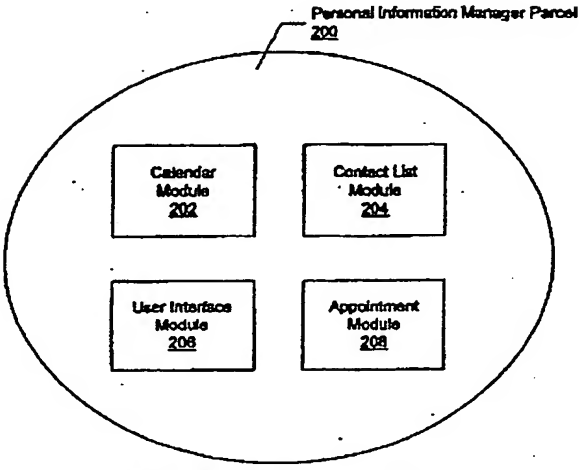
【図１２】



１７０　中央フレームワーク・インスタンス
１７２　ルックアップ
１７４　ロードおよびアンロード
１７６　リフレクション
１７８　公開されたアクセス可能なメソッド
１８０　フレームワーク・メタデータ
１８２　ロードしたモジュールのリスト
Legend　凡例
Read/Write　読み書き
Read only　読み取り専用
Cade　コード
Data　データ

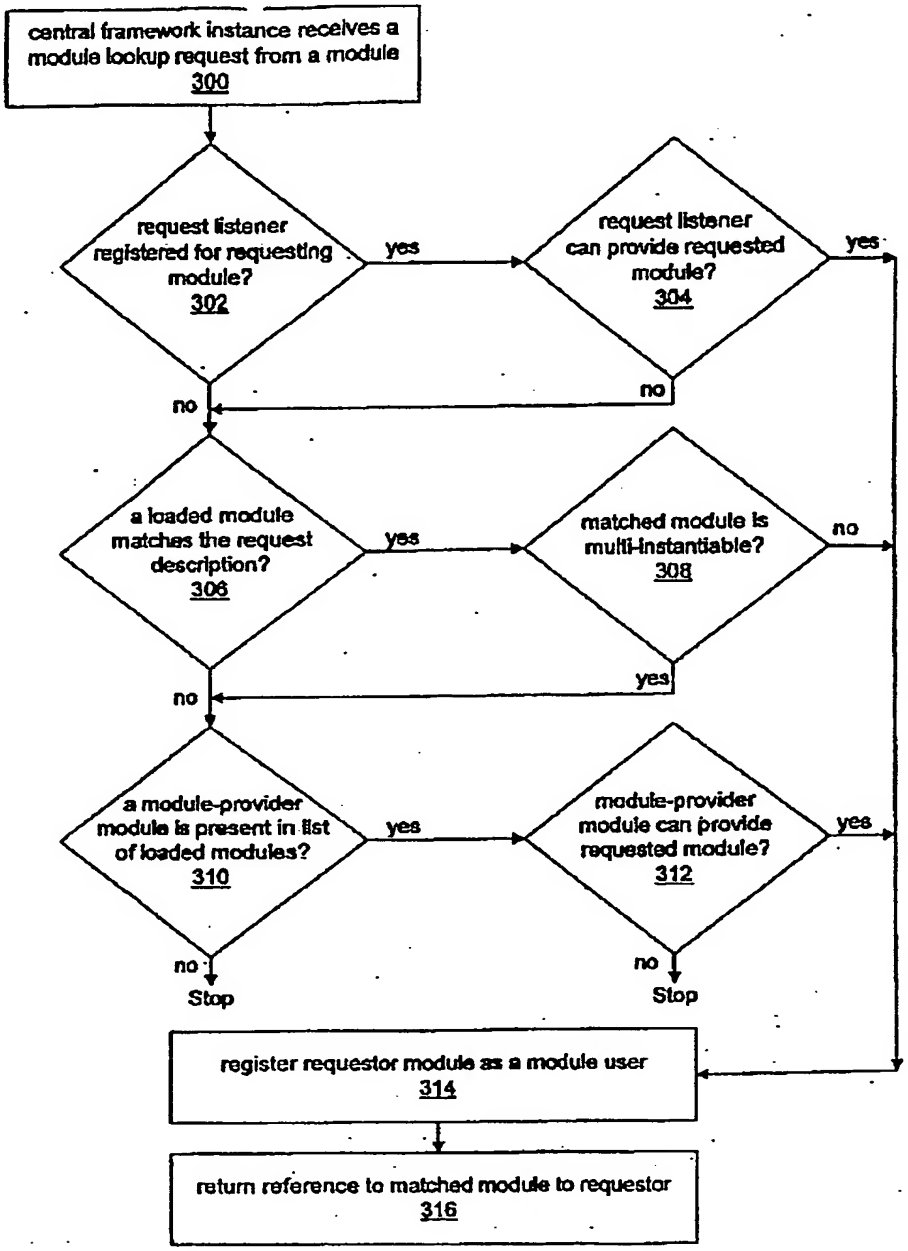


【図１５】



２００　個人情報マネージャ・パーセル
２０２　カレンダ・モジュール
２０４　コンタクト・リスト・モジュール
２０６　ユーザ・インタフェース・モジュール
２０８　アポイントメント・モジュール

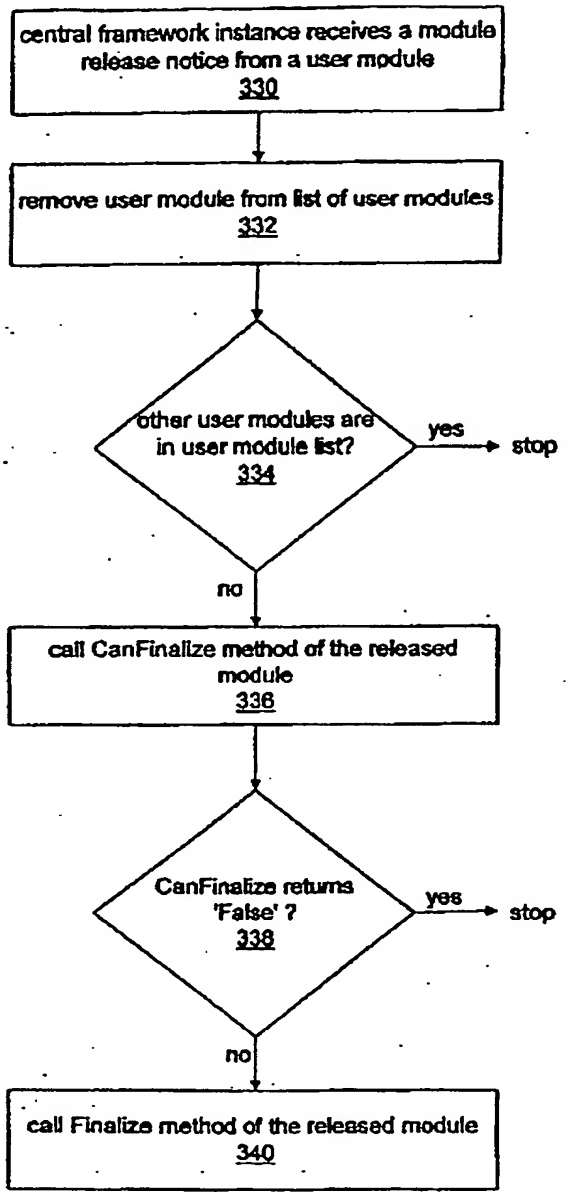
【図１６】



３００　中央フレームワーク・インスタンスがモジュール・ルックアップ要求をモジュールから受け取る
３０２　要求リスナが要求モジュールについて登録されているか
３０４　要求リスナが、要求されたモジュールを提供できるか
３０６　ロードされたモジュールが要求記述に合致するか
３０８　合致したモジュールがマルチ・インスタンス化可能か
３１０　モジュール提供側モジュールが、ロードされたモジュールのリスト内に存在するか
３１２　モジュール提供側モジュールが、要求されたモジュールを提供できるか
３１４　要求側モジュールをモジュール・ユーザとして登録する
３１６　合致したモジュールへの参照を要求側へ返す

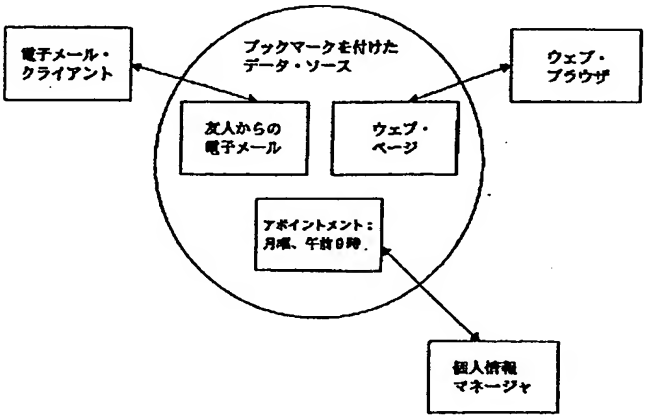
【図１７】

３３０　中央フレームワーク・インスタンスがモジュール解放通知をユーザ・モジュールから受け取る
３３２　ユーザ・モジュールをユーザ・モジュールのリストから削除する
３３４　他のユーザ・モジュールがユーザ・モジュール・リスト内に存在するか
３３６　解放されるモジュールのＣａｎＦｉｎａｌｉｚｅメソッドを呼び出す
３３８　ＣａｎＦｉｎａｌｉｚｅが「Ｆａｌｓｅ」を返すか
３４０　解放されるモジュールのＦｉｎａｌｉｚｅメソッドを呼び出す

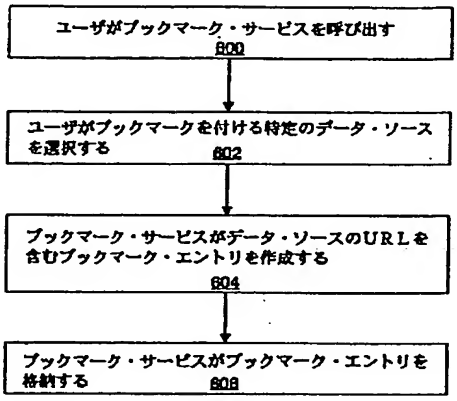


【手続補正書】
【提出日】平成１２年１０月１３日（２０００．１０．１３）
【手続補正１】
【補正対象書類名】図面
【補正対象項目名】全図
【補正方法】変更
【補正内容】

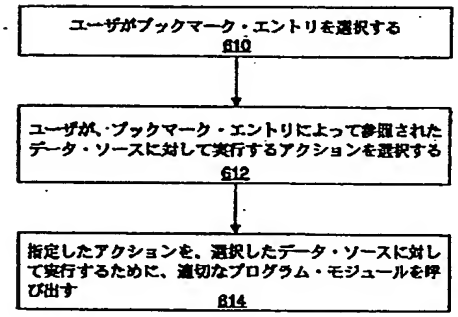
【図１】


電子メール・
クライアント
ブックマークを付けた
データ・ソース
ウェブ・
ブラウザ
友人からの
電子メール
ウェブ・
ページ
アポイントメント：
月曜、午前８時
個人情報
マネージャ

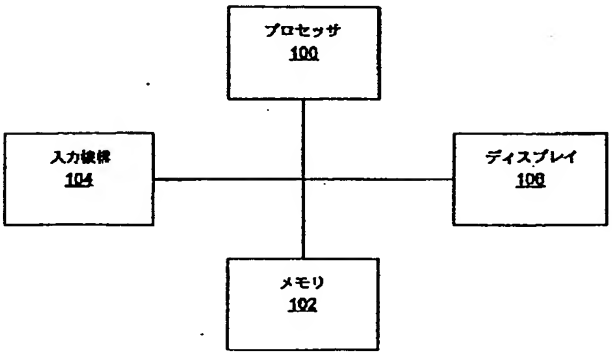
【図２】


ユーザがブックマーク・サービスを呼び出す
600
ユーザがブックマークを付ける特定のデータ・ソース
を選択する
602
ブックマーク・サービスがデータ・ソースのＵＲＬを
含むブックマーク・エントリを作成する
604
ブックマーク・サービスがブックマーク・エントリを
格納する
608

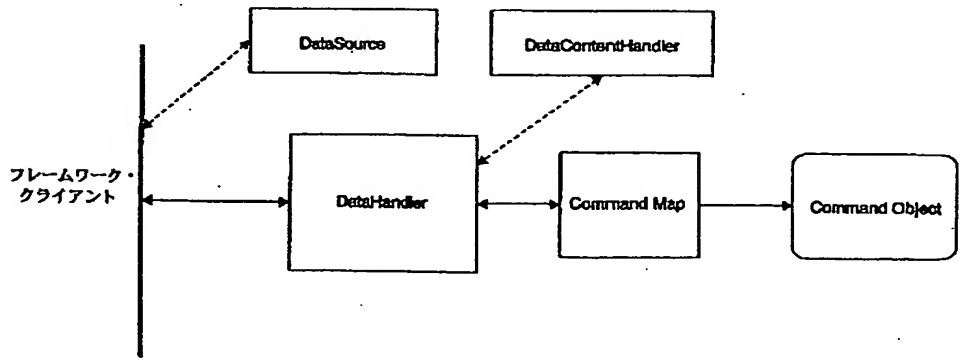
【図３】


ユーザがブックマーク・エントリを選択する
610
ユーザが、ブックマーク・エントリによって参照された
データ・ソースに対して実行するアクションを選択する
612
指定したアクションを、選択したデータ・ソースに対し
て実行するために、適切なプログラム・モジュールを呼
び出す
614

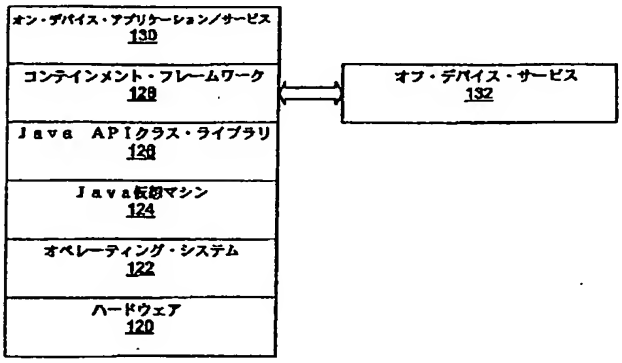
【図５】


プロセッサ
100
入力装置
104
ディスプレイ
106
メモリ
102

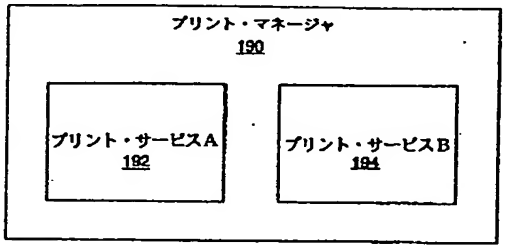
【図４】


DataSource
DataContentHandler
フレームワーク・
クライアント
DataHandler
Command Map
Command Object

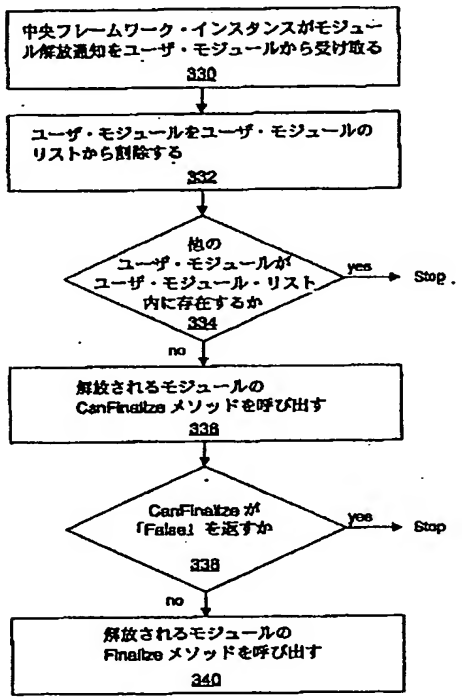
【図６】



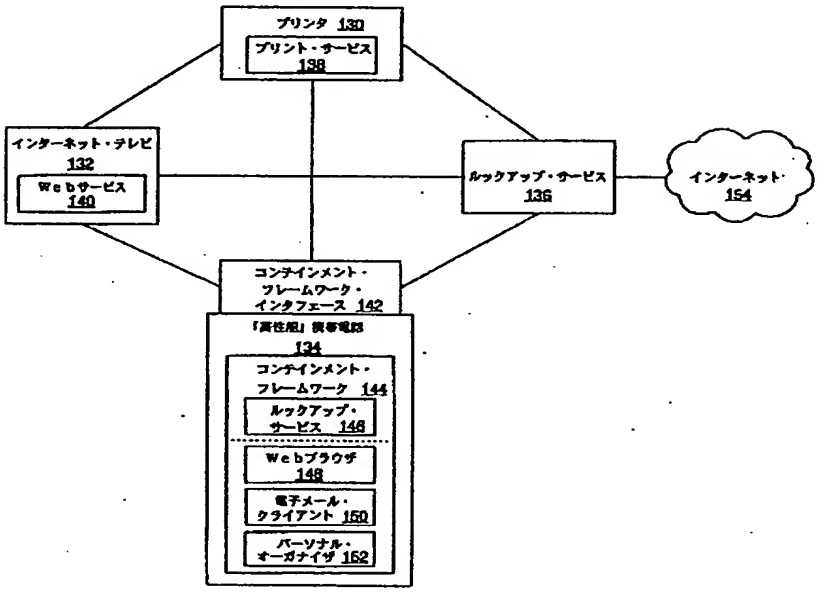
【図１３】



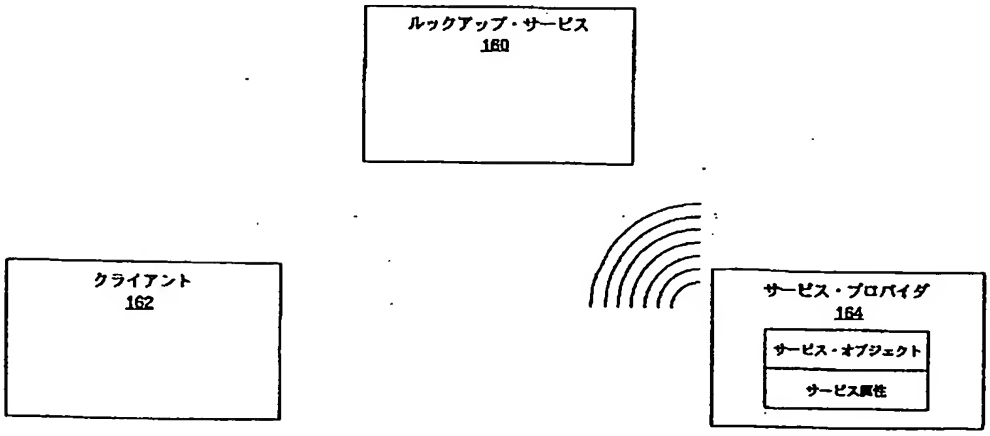
【図１７】



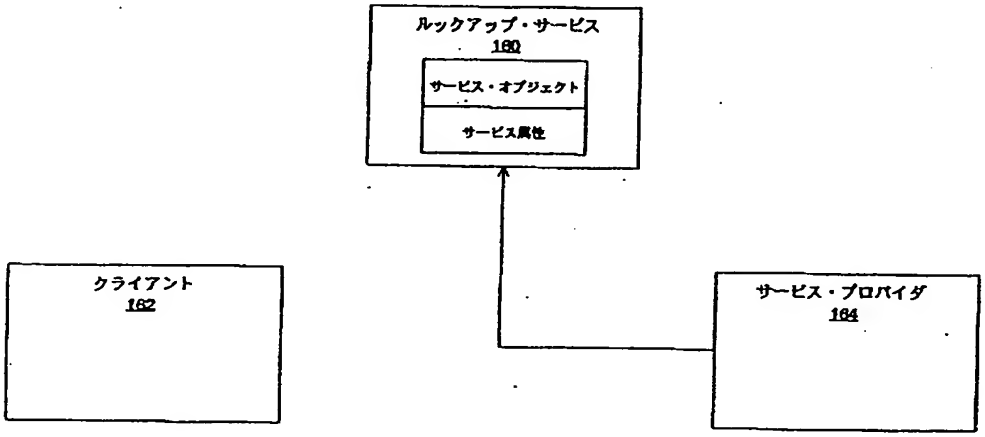
【図７】



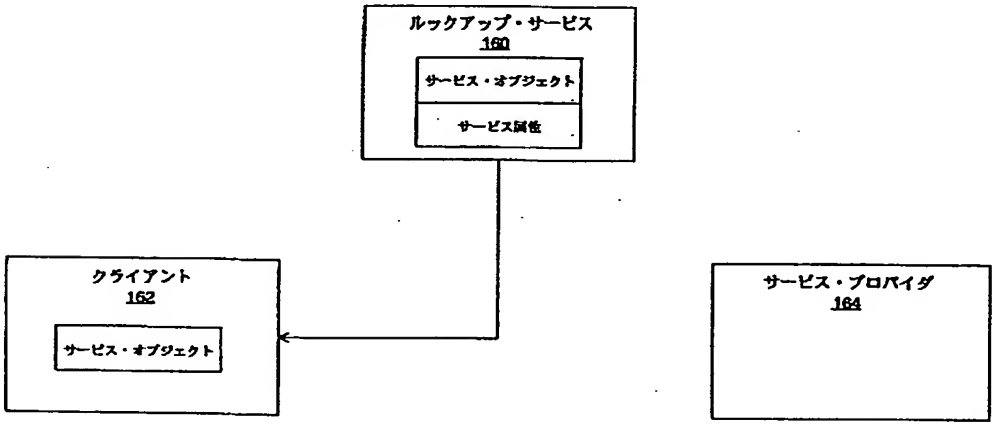
【図８】



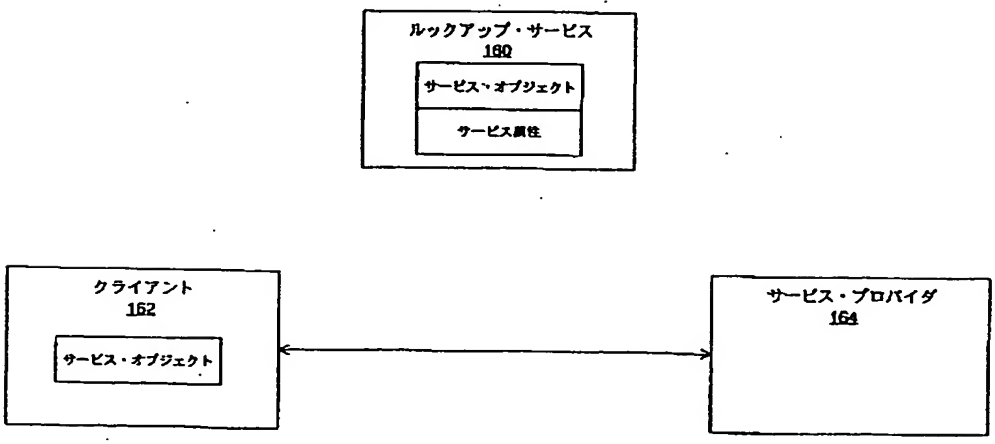
【図９】


ルックアップ・サービス
160
サービス・オブジェクト
サービス属性
クライアント
162
サービス・プロバイダ
164

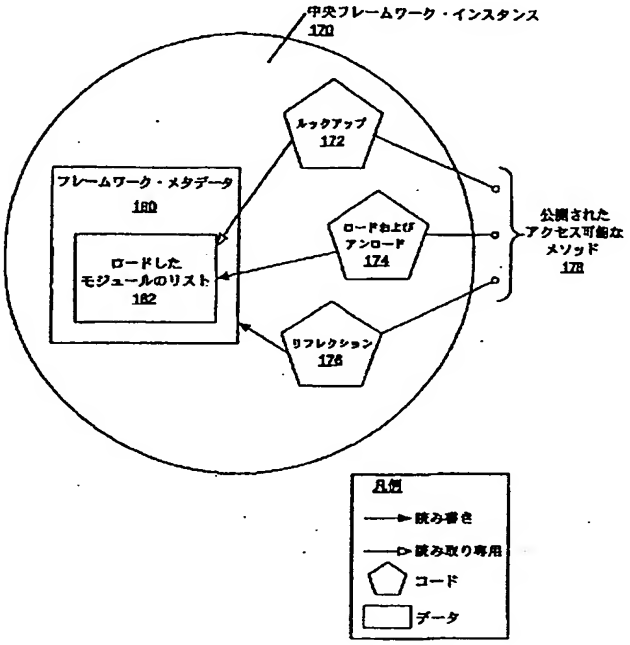
【図１０】


ルックアップ・サービス
160
サービス・オブジェクト
サービス属性
クライアント
162
サービス・オブジェクト
サービス・プロバイダ
164

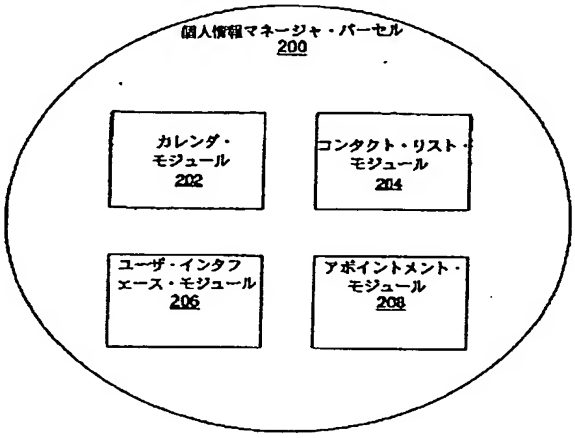
【図１１】


ルックアップ・サービス
160
サービス・オブジェクト
サービス属性
クライアント
162
サービス・オブジェクト
サービス・プロバイダ
164

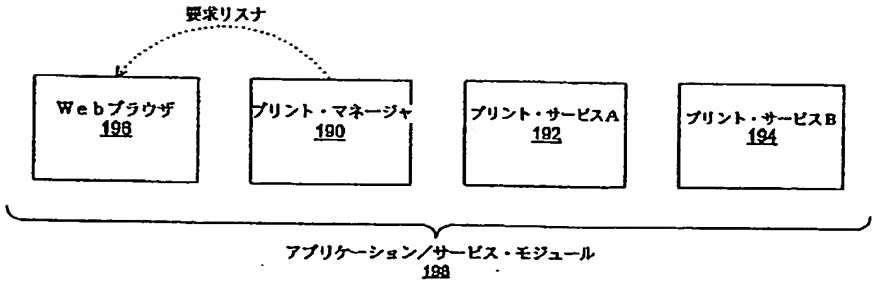
【図１２】


中央フレームワーク・インスタンス
170
ルックアップ
172
フレームワーク・メタデータ
180
ロードした
モジュールのリスト
182
ロードおよび
アンロード
174
リフレクション
176
公開された
アクセス可能な
メソッド
178
凡例
読み書き
読み取り専用
コード
データ

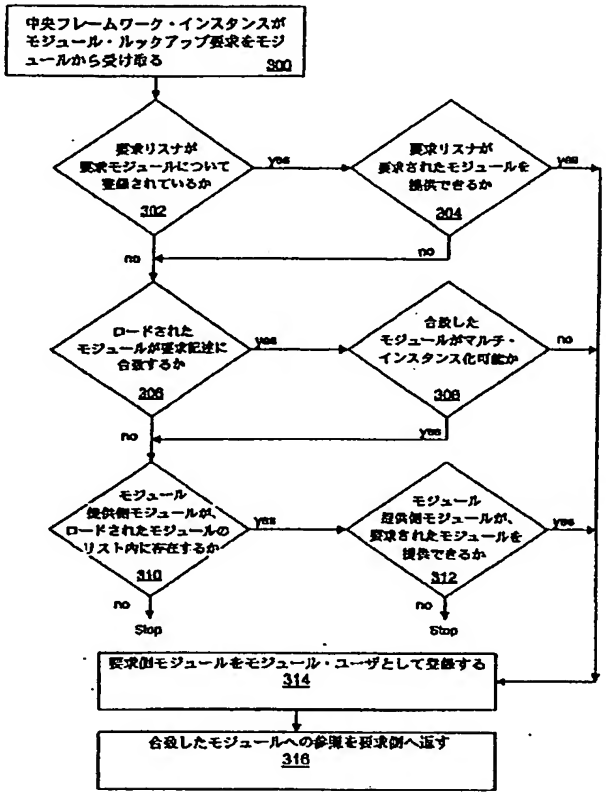
【図１５】


個人情報マネージャ・パーセル
200
カレンダ・
モジュール
202
コンタクト・リスト・
モジュール
204
ユーザ・インタフ
ェース・モジュール
206
アポイントメント・
モジュール
208

【図１４】


要求リスナ
Webブラウザ
196
プリント・マネージャ
190
プリント・サービスA
192
プリント・サービスB
194
アプリケーション／サービス・モジュール
198

【図１６】



フロントページの続き

(72)発明者　マーク・ベイカー
カナダ国・ケイ１エム　２シイ５・オンタリオ州・オタワ・マッケイ　ストリート・399